Lecture 4, April 6

ECS 235B, Foundations of Computer and Information Security

Spring Quarter 2026

Expressive Power

- How do the sets of systems that models can describe compare?
  - If HRU equivalent to SPM, SPM provides more specific answer to safety question
  - If HRU describes more systems, SPM applies only to the systems it can describe

HRU *vs*. SPM

- SPM more abstract
  - Analyses focus on limits of model, not details of representation
- HRU allows revocation
  - SPM has no equivalent to delete, destroy
- HRU allows multiparent creates
  - SPM cannot express multiparent creates easily, and not at all if the parents are of different types because *can*•*create* allows for only one type of creator

Multiparent Create

- Solves mutual suspicion problem
  - Create proxy jointly, each gives it needed rights
- In HRU:

```
command multicreate(s0, s1, o)
if m in a[s0, s1] and m in a[s1, s0]
then
  create object o;
  enter r into a[s0, o];
  enter r into a[s1, o];
end
```

SPM and Multiparent Create

- Called *Extended SPM* or *ESPM*
- *cc* extended in obvious way
  - $cc \subseteq TS \times \ldots \times TS \times T$
- Symbols
  - $\mathbf{X}_1, \ldots, \mathbf{X}_n$ parents, $\mathbf{Y}$ created
  - $R_{1,i}, R_{2,i}, R_3, R_{4,i} \subseteq R$
- Rules
  - $cr_{P,i}(\tau(\mathbf{X}_1), \ldots, \tau(\mathbf{X}_n)) = \mathbf{Y}/R_{1,1} \cup \mathbf{X}_i/R_{2,i}$
  - $cr_C(\tau(\mathbf{X}_1), \ldots, \tau(\mathbf{X}_n)) = \mathbf{Y}/R_3 \cup \mathbf{X}_1/R_{4,1} \cup \ldots \cup \mathbf{X}_n/R_{4,n}$

Example

- Anna, Bill must do something cooperatively
  - But they don't trust each other
- Jointly create a proxy
  - Each gives proxy only necessary rights
- In ESPM:
  - Anna, Bill type a; proxy type p; right $x \in R$
  - $cc(a, a) = p$
  - $cr_{\text{Anna}}(a, a, p) = cr_{\text{Bill}}(a, a, p) = \varnothing$
  - $cr_{\text{proxy}}(a, a, p) = \{ \text{Anna}/x, \text{Bill}/x \}$

2-Parent Joint Create Suffices

- Goal: emulate 3-parent joint create with 2-parent joint create
- Definition of 3-parent joint create (subjects $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$; child $\mathbf{C}$):
  - $cc(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = Z \subseteq T$
  - $cr_{\mathbf{P}1}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{1,1} \cup \mathbf{P}_1/R_{2,1}$
  - $cr_{\mathbf{P}2}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{1,2} \cup \mathbf{P}_2/R_{2,2}$
  - $cr_{\mathbf{P}3}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{1,3} \cup \mathbf{P}_3/R_{2,3}$

General Approach

- Define agents for parents and child
  - Agents act as surrogates for parents
  - If create fails, parents have no extra rights
  - If create succeeds, parents, child have exactly same rights as in 3-parent creates
    - Only extra rights are to agents (which are never used again, and so these rights are irrelevant)

Entities and Types

- Parents $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ have types p_1, p_2, p_3
- Child **C** of type c
- Parent agents $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$ of types a_1, a_2, a_3
- Child agent **S** of type s
- Type t is parentage
  - if **X**/$t \in dom$(**Y**), **X** is **Y**'s parent
- Types t, a_1, a_2, a_3, s are new types

can•create

- Following added to *can•create*:
  - cc(p_1) = a_1
  - cc(p_2, a_1) = a_2
  - cc(p_3, a_2) = a_3
    - Parents creating their agents; note agents have maximum of 2 parents
  - cc(a_3) = s
    - Agent of all parents creates agent of child
  - cc(s) = c
    - Agent of child creates child

Creation Rules

- Following added to create rule:
  - $cr_P(p_1, a_1) = \varnothing$
  - $cr_C(p_1, a_1) = p_1/Rtc$
    - Agent's parent set to creating parent; agent has all rights over parent
  - $cr_{Pfirst}(p_2, a_1, a_2) = \varnothing$
  - $cr_{Psecond}(p_2, a_1, a_2) = \varnothing$
  - $cr_C(p_2, a_1, a_2) = p_2/Rtc \cup a_1/tc$
    - Agent's parent set to creating parent and agent; agent has all rights over parent (but not over agent)

Creation Rules

- $cr_{Pfirst}(p_3, a_2, a_3) = \varnothing$
- $cr_{Psecond}(p_3, a_2, a_3) = \varnothing$
- $cr_C(p_3, a_2, a_3) = p_3/Rtc \cup a_2/tc$
  - Agent's parent set to creating parent and agent; agent has all rights over parent (but not over agent)
- $cr_P(a_3, s) = \varnothing$
- $cr_C(a_3, s) = a_3/tc$
  - Child's agent has third agent as parent $cr_P(a_3, s) = \varnothing$
- $cr_P(s, c) = \mathbf{C}/Rtc$
- $cr_C(s, c) = c/R_3t$
  - Child's agent gets full rights over child; child gets R_3 rights over agent

Link Predicates

- Idea: no tickets to parents until child created
  - Done by requiring each agent to have its own parent rights
  - $link_1(\mathbf{A}_2, \mathbf{A}_1) = \mathbf{A}_1/t \in dom(\mathbf{A}_2) \wedge \mathbf{A}_2/t \in dom(\mathbf{A}_2)$
  - $link_1(\mathbf{A}_3, \mathbf{A}_2) = \mathbf{A}_2/t \in dom(\mathbf{A}_3) \wedge \mathbf{A}_3/t \in dom(\mathbf{A}_3)$
  - $link_2(\mathbf{S}, \mathbf{A}_3) = \mathbf{A}_3/t \in dom(\mathbf{S}) \wedge \mathbf{C}/t \in dom(\mathbf{C})$
  - $link_3(\mathbf{A}_1, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_1)$
  - $link_3(\mathbf{A}_2, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_2)$
  - $link_3(\mathbf{A}_3, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_3)$
  - $link_4(\mathbf{A}_1, \mathbf{P}_1) = \mathbf{P}_1/t \in dom(\mathbf{A}_1) \wedge \mathbf{A}_1/t \in dom(\mathbf{A}_1)$
  - $link_4(\mathbf{A}_2, \mathbf{P}_2) = \mathbf{P}_2/t \in dom(\mathbf{A}_2) \wedge \mathbf{A}_2/t \in dom(\mathbf{A}_2)$
  - $link_4(\mathbf{A}_3, \mathbf{P}_3) = \mathbf{P}_3/t \in dom(\mathbf{A}_3) \wedge \mathbf{A}_3/t \in dom(\mathbf{A}_3)$

Filter Functions

- $f_1(a_2, a_1) = a_1/t \cup c/Rtc$
- $f_1(a_3, a_2) = a_2/t \cup c/Rtc$
- $f_2(s, a_3) = a_3/t \cup c/Rtc$
- $f_3(a_1, c) = p_1/R_{4,1}$
- $f_3(a_2, c) = p_2/R_{4,2}$
- $f_3(a_3, c) = p_3/R_{4,3}$
- $f_4(a_1, p_1) = c/R_{1,1} \cup p_1/R_{2,1}$
- $f_4(a_2, p_2) = c/R_{1,2} \cup p_2/R_{2,2}$
- $f_4(a_3, p_3) = c/R_{1,3} \cup p_3/R_{2,3}$

Construction

Create $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, **S**, **C**; then

- $\mathbf{P}_1$ has no relevant tickets
- $\mathbf{P}_2$ has no relevant tickets
- $\mathbf{P}_3$ has no relevant tickets
- $\mathbf{A}_1$ has $\mathbf{P}_1/Rtc$
- $\mathbf{A}_2$ has $\mathbf{P}_2/Rtc \cup \mathbf{A}_1/tc$
- $\mathbf{A}_3$ has $\mathbf{P}_3/Rtc \cup \mathbf{A}_2/tc$
- **S** has $\mathbf{A}_3/tc \cup \mathbf{C}/Rtc$
- **C** has $\mathbf{C}/R_3t$

Construction

- Only $link_2(\mathbf{S}, \mathbf{A}_3)$ true $\Rightarrow$ apply f_2
  - $\mathbf{A}_3$ has $\mathbf{P}_3/Rtc \cup \mathbf{A}_2/t \cup \mathbf{A}_3/t \cup \mathbf{C}/Rtc$
- Now $link_1(\mathbf{A}_3, \mathbf{A}_2)$ true $\Rightarrow$ apply f_1
  - $\mathbf{A}_2$ has $\mathbf{P}_2/Rtc \cup \mathbf{A}_1/tc \cup \mathbf{A}_2/t \cup \mathbf{C}/Rtc$
- Now $link_1(\mathbf{A}_2, \mathbf{A}_1)$ true $\Rightarrow$ apply f_1
  - $\mathbf{A}_1$ has $\mathbf{P}_2/Rtc \cup \mathbf{A}_1/t \cup \mathbf{C}/Rtc$
- Now all $link_3$s true $\Rightarrow$ apply f_3
  - $\mathbf{C}$ has $\mathbf{C}/R_3 \cup \mathbf{P}_1/R_{4,1} \cup \mathbf{P}_2/R_{4,2} \cup \mathbf{P}_3/R_{4,3}$

Finish Construction

- Now $link_4$ is true $\Rightarrow$ apply f_4
  - $\mathbf{P}_1$ has $\mathbf{C}/R_{1,1} \cup \mathbf{P}_1/R_{2,1}$
  - $\mathbf{P}_2$ has $\mathbf{C}/R_{1,2} \cup \mathbf{P}_2/R_{2,2}$
  - $\mathbf{P}_3$ has $\mathbf{C}/R_{1,3} \cup \mathbf{P}_3/R_{2,3}$
- 3-parent joint create gives same rights to $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, $\mathbf{C}$
- If create of $\mathbf{C}$ fails, $link_2$ fails, so construction fails

Theorem

- The two-parent joint creation operation can implement an *n*-parent joint creation operation with a fixed number of additional types and rights, and augmentations to the link predicates and filter functions.
- **Proof**: by construction, as above
  - Difference is that the two systems need not start at the same initial state

Theorems

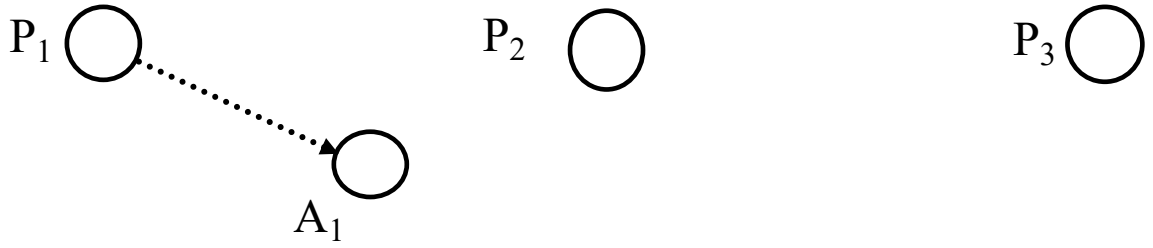
- Monotonic ESPM and the monotonic HRU model are equivalent.
- Safety question in ESPM also decidable if acyclic attenuating scheme
  - Proof similar to that for SPM

Expressiveness

- Graph-based representation to compare models
- Graph
  - Vertex: represents entity, has static type
  - Edge: represents right, has static type
- Graph rewriting rules:
  - *Initial state operations* create graph in a particular state
  - *Node creation operations* add nodes, incoming edges
  - *Edge adding operations* add new edges between existing vertices

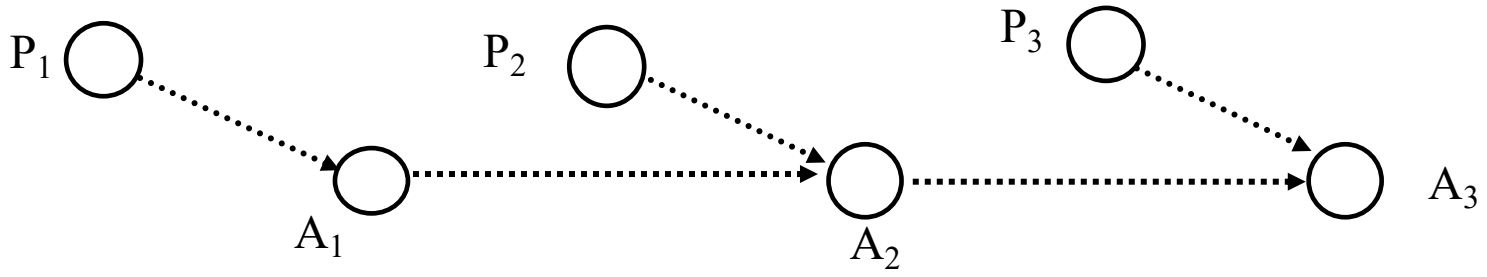
Example: 3-Parent Joint Creation

- Simulate with 2-parent
  - Nodes $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ parents
  - Create node **C** with type *c* with edges of type *e*
  - Add node $\mathbf{A}_1$ of type *a* and edge from $\mathbf{P}_1$ to $\mathbf{A}_1$ of type *e'*



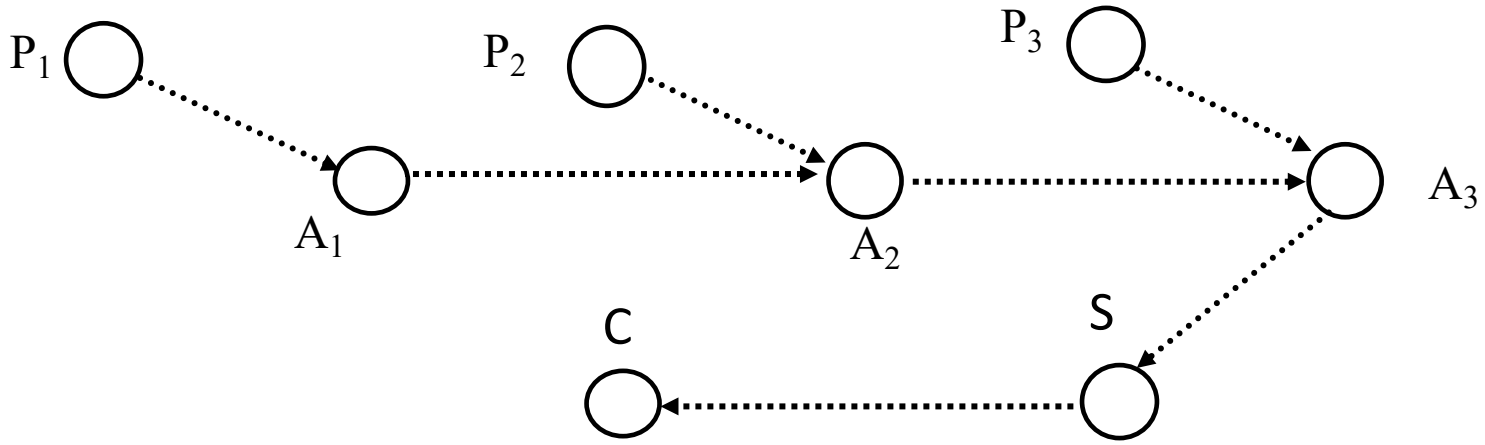
Next Step

- $\mathbf{A}_1$, $\mathbf{P}_2$ create $\mathbf{A}_2$; $\mathbf{A}_2$, $\mathbf{P}_3$ create $\mathbf{A}_3$
- Type of nodes, edges are *a* and *e*′



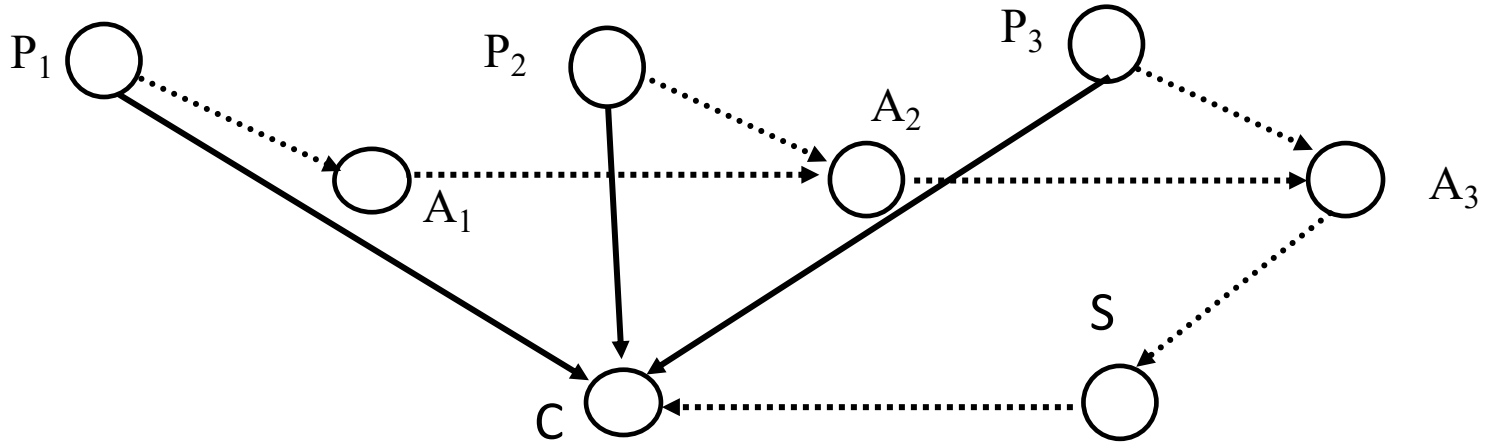
Next Step

- **A_3** creates **S**, of type *a*
- **S** creates **C**, of type *c*



Last Step

- Edge adding operations:
  - $\mathbf{P}_1 \rightarrow \mathbf{A}_1 \rightarrow \mathbf{A}_2 \rightarrow \mathbf{A}_3 \rightarrow \mathbf{S} \rightarrow \mathbf{C}$: $\mathbf{P}_1$ to **C** edge type *e*
  - $\mathbf{P}_2 \rightarrow \mathbf{A}_2 \rightarrow \mathbf{A}_3 \rightarrow \mathbf{S} \rightarrow \mathbf{C}$: $\mathbf{P}_2$ to **C** edge type *e*
  - $\mathbf{P}_3 \rightarrow \mathbf{A}_3 \rightarrow \mathbf{S} \rightarrow \mathbf{C}$: $\mathbf{P}_3$ to **C** edge type *e*



Definitions

- *Scheme*: graph representation as above
- *Model*: set of schemes
- Schemes *A*, *B correspond* if graph for both is identical when all nodes with types not in *A* and edges with types in *A* are deleted

Example

- Above 2-parent joint creation simulation in scheme *TWO*
- Equivalent to 3-parent joint creation scheme *THREE* in which $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, **C** are of same type as in *TWO*, and edges from $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ to **C** are of type *e*, and no types *a* and *e´* exist in *TWO*

Simulation

Scheme *A* simulates scheme *B* iff

- every state *B* can reach has a corresponding state in *A* that *A* can reach; and
- every state that *A* can reach either corresponds to a state *B* can reach, or has a successor state that corresponds to a state *B* can reach
  - The last means that *A* can have intermediate states not corresponding to states in *B*, like the intermediate ones in *TWO* in the simulation of *THREE*

Expressive Power

- If there is a scheme in *MA* that no scheme in *MB* can simulate, *MB less expressive than MA*
- If every scheme in *MA* can be simulated by a scheme in *MB*, *MB as expressive as MA*
- If *MA* as expressive as *MB* and *vice versa*, *MA* and *MB equivalent*

Example

- Scheme *A* in model *M*
  - Nodes $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$
  - 2-parent joint create
  - 1 node type, 1 edge type
  - No edge adding operations
  - Initial state: $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, no edges
- Scheme *B* in model *N*
  - All same as *A* except no 2-parent joint create
  - 1-parent create
- Which is more expressive?

Can *A* Simulate *B*?

- Scheme *A* simulates 1-parent create: have both parents be same node
  - Model *M* as expressive as model *N*

Can *B* Simulate *A*?

- Suppose $\mathbf{X}_1$, $\mathbf{X}_2$ jointly create **Y** in *A*
  - Edges from $\mathbf{X}_1$, $\mathbf{X}_2$ to **Y**, no edge from $\mathbf{X}_3$ to **Y**
- Can *B* simulate this?
  - Without loss of generality, $\mathbf{X}_1$ creates **Y**
  - Must have edge adding operation to add edge from $\mathbf{X}_2$ to **Y**
  - One type of node, one type of edge, so operation can add edge between any 2 nodes

No

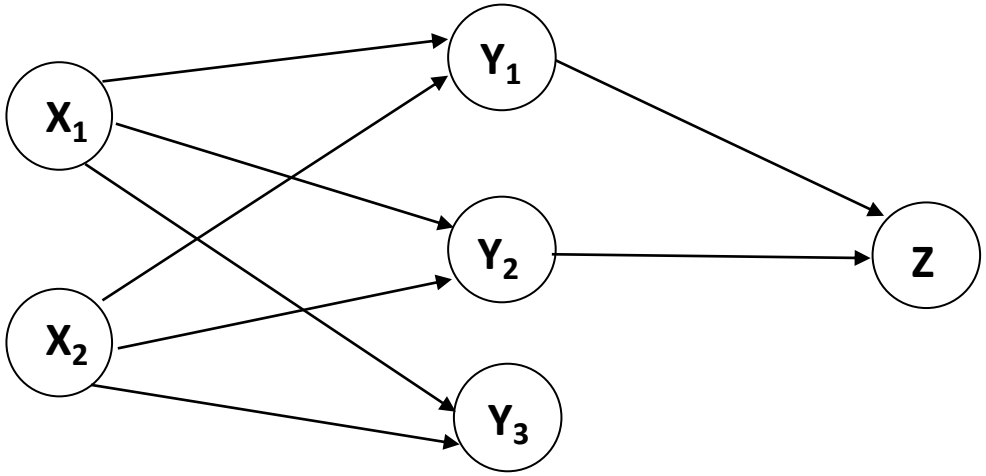
- All nodes in *A* have even number of incoming edges
  - 2-parent create adds 2 incoming edges
- Edge adding operation in *B* that can edge from $\mathbf{X}_2$ to **C** can add one from $\mathbf{X}_3$ to **C**
  - *A* cannot enter this state
  - *B* cannot transition to a state in which **Y** has even number of incoming edges
    - No remove rule
- So *B* cannot simulate *A*; *N* less expressive than *M*

Theorem

- Monotonic single-parent models are less expressive than monotonic multiparent models
- Proof by contradiction
  - Scheme *A* is multiparent model
  - Scheme *B* is single parent create
  - Claim: *B* can simulate *A*, without assumption that they start in the same initial state
    - Note: example assumed same initial state

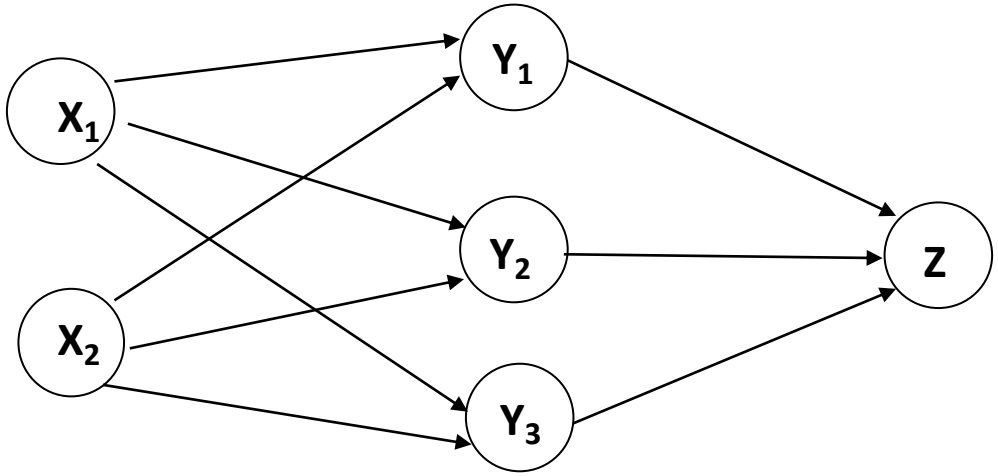
Outline of Proof

- $\mathbf{X}_1$, $\mathbf{X}_2$ nodes in *A*
  - They create $\mathbf{Y}_1$, $\mathbf{Y}_2$, $\mathbf{Y}_3$ using multiparent create rule
  - $\mathbf{Y}_1$, $\mathbf{Y}_2$ create **Z**, again using multiparent create rule
  - *Note*: no edge from $\mathbf{Y}_3$ to **Z** can be added, as *A* has no edge-adding operation



Outline of Proof

- **W**, $\mathbf{X}_1$, $\mathbf{X}_2$ nodes in *B*
  - **W** creates $\mathbf{Y}_1$, $\mathbf{Y}_2$, $\mathbf{Y}_3$ using single parent create rule, and adds edges for $\mathbf{X}_1$, $\mathbf{X}_2$ to all using edge adding rule
  - $\mathbf{Y}_1$ creates **Z**, again using single parent create rule; now must add edge from $\mathbf{Y}_2$ to **Z** to simulate *A*
  - Use same edge adding rule to add edge from $\mathbf{Y}_3$ to **Z**: cannot duplicate this in scheme *A*!



Meaning

- Scheme *B* cannot simulate scheme *A*, contradicting hypothesis
- ESPM more expressive than SPM
  - ESPM multiparent and monotonic
  - SPM monotonic but single parent

Typed Access Matrix Model

- Like ACM, but with set of types *T*
  - All subjects, objects have types
  - Set of types for subjects *TS*
- Protection state is (*S*, *O*, τ, *A*)
  - τ: $O \rightarrow T$ specifies type of each object
  - If **X** subject, τ(**X**) in *TS*
  - If **X** object, τ(**X**) in $T - TS$

Create Rules

- Subject creation
  - **create subject** *s* **of type** *ts*
  - *s* must not exist as subject or object when operation executed
  - $ts \in TS$
- Object creation
  - **create object** *o* **of type** *to*
  - *o* must not exist as subject or object when operation executed
  - $to \in T - TS$

Create Subject

- Precondition: $s \notin S$
- Primitive command: **create subject** s **of type** t
- Postconditions:
  - $S' = S \cup \{ s \}$, $O' = O \cup \{ s \}$
  - $(\forall y \in O)[\tau'(y) = \tau(y)]$, $\tau'(s) = t$
  - $(\forall y \in O')[a'[s, y] = \varnothing]$, $(\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Create Object

- Precondition: $o \notin O$
- Primitive command: **create object** o **of type** t
- Postconditions:
  - $S' = S$, $O' = O \cup \{ o \}$
  - $(\forall y \in O)[\tau'(y) = \tau(y)]$, $\tau'(o) = t$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

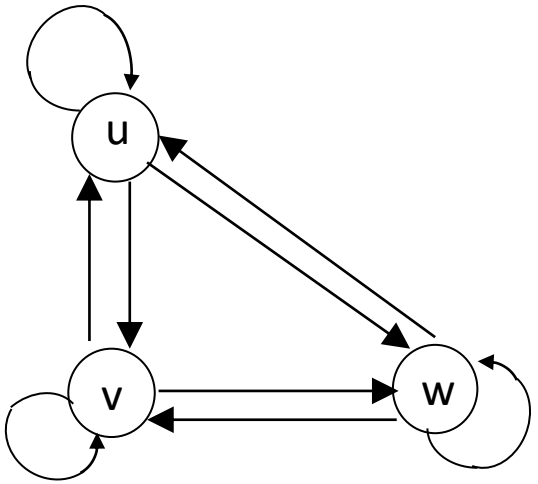
Definitions

- MTAM Model: TAM model without **delete**, **destroy**
  - MTAM is Monotonic TAM
- $\alpha(x_1{:}t_1, \ldots, x_n{:}t_n)$ create command
  - t_i child type in α if any of **create subject** x_i **of type** t_i or **create object** x_i **of type** t_i occur in α
  - t_i parent type otherwise

Cyclic Creates

command *cry•havoc*(s_1:*u*, s_2:*u*, o_1:*v*, o_2:*v*, o_3:*w*, o_4:*w*)
 create subject s_1 **of type** *u*;
 create object o_1 **of type** *v*;
 create object o_3 **of type** *w*;
 enter *r* **into** *a*[s_2, s_1];
 enter *r* **into** *a*[s_2, o_2];
 enter *r* **into** *a*[s_2, o_4]
end

Creation Graph

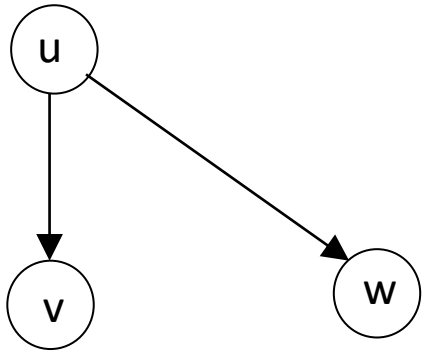


- *u*, *v*, *w* child types
- *u*, *v*, *w* also parent types
- Graph: lines from parent types to child types
- This one has cycles

Acyclic Creates

```
command cry•havoc(s1:u, s2:u, o1:v, o3:w)
 create object o1 of type v;
 create object o3 of type w;
 enter r into a[s2, s1];
 enter r into a[s2, o1];
 enter r into a[s2, o3]
end
```

Creation Graph



- *v*, *w* child types
- *u* parent type
- Graph: lines from parent types to child types
- This one has no cycles

Theorems

- Safety decidable for systems with acyclic MTAM schemes
  - In fact, it's *NP-hard*
- Safety for acyclic ternary MATM decidable in time polynomial in the size of initial ACM
  - "Ternary" means commands have no more than 3 parameters
  - Equivalent in expressive power to MTAM

Security Properties

- Question: given two models, do they have the same security properties?
  - First comes theory
  - Then comes an example comparison
- Basic idea: view access request as query asking if subject has right to perform action on object
  - If both always give the same answer, they have the same security properties

Alternate Definition of "Scheme"

- Σ set of states
- *Q* set of queries
- *e*: Σ × *Q* → { *true*, *false* }
  - Called *entailment relation*
- *T* set of state transition rules
- (Σ, *Q*, *e*, *T*) is an *access control scheme*

Alternate Definition of "Scheme"

- *s* tries to access *o*
  - Corresponds to query $q \in Q$
- If state $\sigma \in \Sigma$ allows access, then $e(\sigma, q)$ *true*; otherwise, $e(\sigma, q)$ *false*
- Write change of state from σ_0 to σ_1 as $\sigma_0 \mapsto \sigma_1$
  - Emphasizing we're looking at *permissions*
  - Multiple transitions are $\sigma_0 \mapsto_\tau^* \sigma_n$
  - σ_n said to be *τ-reachable from* σ_0

Example: Take-Grant

- Σ set of all possible protection graphs
- *Q* set of queries

  $\{ can\bullet share(\alpha, \mathbf{v}_1, \mathbf{v}_2, G_0) \mid \alpha \in R, \mathbf{v}_1, \mathbf{v}_2 \in G_0 \}$
- $e(\sigma_0, q)$ = *true* if q ($can\bullet share(\alpha, \mathbf{v}_1, \mathbf{v}_2, G_0)$ for particular $\alpha, \mathbf{v}_1, \mathbf{v}_2$) holds; $e(\sigma_0, q)$ = *false* if not
- *T* set of sequences of take, grant, create, remove rules

Security Analysis Instance

- Let (Σ, *Q*, *e*, *T*) be an *access control scheme*
- Tuple (σ, *q*, τ, Π) is *security analysis instance*, where:
  - $\sigma \in \Sigma$
  - $q \in Q$
  - $\tau \in T$
  - Π is ∀ or ∃
- If Π is ∃, *existential* security analysis
  - Is there a state σ′ such that $\sigma \mapsto_\tau^* \sigma'$, $e(\sigma', q) = true$?
- If Π is ∀, *universal* security analysis
  - For all states σ′ such that $\sigma \mapsto_\tau^* \sigma'$, is $e(\sigma', q) = true$?

Example: Take-Grant

- $\sigma_0 = G_0$
- q is $can \bullet share(r, \mathbf{v}_1, \mathbf{v}_2, G_0)$
- τ is sequence of take-grant rules
- Π is $\exists$
- Security analysis instance examines whether $\mathbf{v}_1$ has r rights over $\mathbf{v}_2$ in graph with initial state G_0
- So safety question is security analysis instance

Comparing Two Models

- Each query in *A* corresponds to a query in *B*
- Each (state, state transition) in *A* corresponds to (state, state transition) in *B*

Formally:

- $A = (\Sigma^A, Q^A, e^A, T^A)$ and $B = (\Sigma^B, Q^B, e^B, T^B)$
- *mapping* from *A* to *B* is:
  - $f : (\Sigma^A \times T^A) \cup Q^A \rightarrow (\Sigma^B \times T^B) \cup Q^B$

Image of Instance

- *f* mapping from A to B
- *image of a security analysis instance:*

  $(\sigma^A, q^A, \tau^A, \Pi)$ under *f* is $(\sigma^B, q^B, \tau^B, \Pi)$,

  where:
  - $f((\sigma^A, \tau^A)) = (\sigma^B, \tau^B)$
  - $f(q^A) = q^B$
- *f* is *security-preserving* if every security analysis instance in *A* is true iff its image is true

Composition of Queries

- Let (Σ, *Q*, *e*, *T*) be an *access control scheme*
- Tuple (σ, *φ*, τ, Π) is compositional *security analysis instance*, where *φ* is propositional logic formula of queries from *Q*
- *image of compositional security analysis instance* defined accordingly
- *f* is *strongly security-preserving* if every compositional security analysis instance in *A* is true iff its image is true

State-Matching Reduction

- $A = (\Sigma^A, Q^A, e^A, T^A)$, $B = (\Sigma^B, Q^B, e^B, T^B)$, f mapping from A to B
- σ^A, σ^B equivalent under the mapping f when $e^A(\sigma^A, q^A) = e^B(\sigma^B, q^B)$
- f state-matching reduction if for all $\sigma^A \in S^A$, $\tau^A \in T^A$, $(\sigma^B, \tau^B) = f((\sigma^A, \tau^A))$ has the following 2 properties:

Property 1

- For every state σ'^{A} in scheme *A* such that $\sigma^{A} \mapsto_{\tau}^{*} \sigma'^{A}$, there is a state σ'^{B} in scheme *B* such that $\sigma^{B} \mapsto_{\tau}^{*} \sigma'^{B}$, and σ'^{A} and σ'^{B} are equivalent under the mapping *f*
  - That is, for every reachable state in *A*, a matching state in *B* gives the same answer for every query

Property 2

- For every state σ'^{B} in scheme *B* such that $\sigma^{B} \mapsto_{\tau}^{*} \sigma'^{B}$, there is a state σ'^{A} in scheme *A* such that $\sigma^{A} \mapsto_{\tau}^{*} \sigma'^{A}$, and σ'^{A} and σ'^{B} are equivalent under the mapping *f*
  - That is, for every reachable state in *B*, a matching state in *A* gives the same answer for every query

Theorem

Mapping *f* from scheme *A* to *B* is strongly security-preserving iff *f* is a state-matching reduction

Proof ($\Longrightarrow$)

- Must show (σ^A, φ^A, τ^A, Π) true iff (σ^B, φ^B, τ^B, Π) true
- Π is $\exists$: assume τ^A-reachable state σ'^A from σ^A in which φ^A true
  - By property 1, there is a state σ'^B corresponding to σ'^A in which φ^B holds
- Π is $\forall$: assume τ^A-reachable state σ'^A from σ^A in which φ^A false
  - By property 1, there is a state σ'^B corresponding to σ'^A in which φ^B false
- Same for φ^B with τ^B-reachable state σ'^B from σ^B
- So (σ^A, φ^A, τ^A, Π) true iff (σ^B, φ^B, τ^B, Π) true

Proof ($\Longleftarrow$)

- Let f be map from A to B but not state-matching reduction. Then there are $\sigma^A \in S^A$, $\tau^A \in T^A$, $(\sigma^B, \tau^B) = f((\sigma^A, \tau^A))$ violating at least one of the properties
- Assume it's property 1; σ^A, σ^B corresponding states. There is a τ^A-reachable state σ'^A from σ^A such that no τ^B-reachable state from σ^B is equivalent to σ'^B

Proof ($\Longleftarrow$)

- Generate φ^A and φ^B such that the existential compositional security analysis in A is true but in B is false
  - To do this, look at each $q^A \in Q^A$
  - If $e(\sigma'^A, q^A)$ *true*, conjoin q^A to φ^A; otherwise, conjoin $\neg q^A$ to φ^A
  - Then $e(\sigma'^A, q^A)$ *true*; but for $\varphi^B = f(\varphi^A)$ and all states σ'^B that are τ^B-reachable from σ^B, $e(\sigma'^B, q^B)$ *false*
- Thus, f is not strongly security-preserving
- Argument for property 2 is similar

Expressive Power

If access control model *MA* has a scheme that cannot be mapped into a scheme in access control model *MB* using a state-matching reduction, then model *MB* is *less expressive than* model *MA*.

If every scheme in model *MA* can be mapped into a scheme in model *MB* using a state-matching reduction, then model *MB* is *as expressive as* model *MA*.

If *MA* is as expressive as *MB*, and MB is as expressive as *MA*, the models are *equivalent*

- Note this does not assume monotonicity, unlike earlier definition

Augmented Typed Access Control Matrix

- Add a test for the *absence* of rights to TAM

```
command add•right(s:u, o:v)
        if own in a[s,o] and r not in a[s,o]
        then
                enter r into a[s,o]
end
```

- How does this affect the answer to the safety question?

Safety Question

- ATAM can be mapped onto TAM
- But will the mapping, or any such mapping, preserve security properties?
- Approach: consider TAM as an access control model

TAM as Access Control Model

- S set of subjects; S_σ subjects in state σ
- O set of objects; O_σ objects in state σ
- R set of rights; R_σ rights in state σ
- T set of types; T_σ subjects in state σ
- $t : S_\sigma \cup O_\sigma \longrightarrow T_\sigma$ gives type of any subject or object
- State σ defined as $(S_\sigma, O_\sigma, R_\sigma, T_\sigma, t)$
- In TAM, query is of form "is $r \in a[s,o]$", and $e(s, r \in a[s,o])$ *true* iff $s \in S_\sigma$, $o \in O_\sigma$, $r \in R_\sigma$, $r \in a_\sigma[s,o]$ are all true

ATAM as Access Control Model

Same as TAM with one addition:

- ATAM also allows queries of form "is $r \notin a[s,o]$", and $e(s, r \notin a[s,o])$ *true* iff $s \in S_\sigma$, $o \in O_\sigma$, $r \in R_\sigma$, $r \notin a_\sigma[s,o]$ are all true

Theorem

A state-matching reduction from ATAM to TAM does not exist.

Outline of proof: by contradiction

- Consider two state transitions, one that creates subject and one that adds right *r* to an element of the matrix
- Can determine an upper bound on the number of answers to TAM query a command can change; bound depends on state and commands

Proof

- Assume *f* is state-matching reduction from ATAM to TAM
- Consider simple ATAM scheme:
  - Initial state σ_0 has no subjects, objects
  - All entities have type *t*
  - Only one right *r*
  - Query $q_{ij} = r \in a[s,o]$; query $\underline{q_{ij}} = r \notin a[s,o]$
  - 2 state transition rules:
    - *make•subj*(*s* : *t*) creates subject *s* of type *t*
    - *add•right*(*x* : *t*, *y* : *t*) adds right *r* to *a*[*x*, *y*]

Proof

- TAM: superscript *T* represents components of that system
  - So initial state is $\sigma_0{}^T = f(\sigma_0)$, transitions are $\tau^T = f(\tau)$
- By definition of state-matching reduction, how *f* maps queries does not depend on initial state or state transitions of a model
- Let *p*, *q* be queries in ATAM and p^T, q^T the corresponding queries in TAM; if $p \neq q$, then $p^T \neq q^T$
- As commands in TAM execute, they can change the value (response) of q_{ij}
- Upper bound on the number of values of queries a single command can change is *m* (number of `enter` or `add•right` operations)

Proof

- Choose $n > m$
- In ATAM, construct state σ_k such that:
  - $\sigma_0 \longrightarrow^* \sigma_k$; and
  - $e(\sigma_k, \neg q_{1,1} \wedge \underline{q_{1,1}} \wedge \ldots \wedge \neg q_{n,n} \wedge \underline{q_{n,n}})$ is true
- So $e(\sigma_k, q_{i,j})$ is *false*, $e(\sigma_k, \underline{q_{i,j}})$ is *true* for all $1 \leq i, j \leq n$
- As f is a state-matching reduction, there is a state $\sigma_k{}^T$ in TAM that causes the corresponding queries to be answered the same way
- Consider $\sigma_0{}^T \longrightarrow \sigma_1{}^T \longrightarrow \ldots \longrightarrow \sigma_k{}^T$; choose first state $\sigma_C{}^T$ such that $e(\sigma_C{}^T, q_{i,j}{}^T \vee \underline{q_{i,j}{}^T})$ is true for all $1 \leq i, j \leq n$

Proof

- In $\sigma_{C\text{-}1}{}^{T}$, $e(\sigma_{C\text{-}1}{}^{T}, q_{v,w}{}^{T} \vee \underline{q_{v,w}{}^{T}})$ is *false* for some $1 \leq v, w \leq n$, so $e(\sigma_{C\text{-}1}{}^{T}, \neg q_{v,w}{}^{T} \wedge \neg \underline{q_{v,w}{}^{T}})$ is *true*
- State σ in ATAM for which $e(\sigma, \neg q_{v,w} \wedge \neg \underline{q_{v,w}})$ is true is one in which either s_v or s_w or both does not exist
- Thus in that state, one of the following 2 queries holds:
  - $Q_1 = \neg q_{v,1} \wedge \neg \underline{q_{v,1}} \wedge \ldots \wedge \neg q_{n,v} \wedge \neg \underline{q_{n,v}}$
  - $Q_1 = \neg q_{w,1} \wedge \neg \underline{q_{w,1}} \wedge \ldots \wedge \neg q_{n,w} \wedge \neg \underline{q_{n,w}}$
- So in TAM, $e(\sigma_{C\text{-}1}{}^{T}, Q_1{}^{T} \wedge Q_2{}^{T})$ is *true*

Proof

- Now consider the transition from σ_{C-1}^{T} to σ_{C}^{T}
- Values of at least n queries in Q_1 or Q_2 must change from false to true
- But each command can change at most $m < n$ queries
- This is a contradiction
- So no such f can exist, proving the result

Thus, ATAM can express security properties that TAM cannot

Security Policy

- Policy partitions system states into:
  - Authorized (secure)
    - These are states the system can enter
  - Unauthorized (nonsecure)
    - If the system enters any of these states, it's a security violation
- Secure system
  - Starts in authorized state
  - Never enters unauthorized state

Confidentiality

- *X* set of entities, *I* information
- *I* has the *confidentiality* property with respect to *X* if no $x \in X$ can obtain information from *I*
- *I* can be disclosed to others
- Example:
  - *X* set of students
  - *I* final exam answer key
  - *I* is confidential with respect to *X* if students cannot obtain final exam answer key

Integrity

- *X* set of entities, *I* information
- *I* has the *integrity* property with respect to *X* if all $x \in X$ trust information in *I*
- Types of integrity:
  - Trust *I*, its conveyance and protection (data integrity)
  - *I* information about origin of something or an identity (origin integrity, authentication)
  - *I* resource: means resource functions as it should (assurance)

Availability

- *X* set of entities, *I* information (or a resource)
- *I* has the *availability* property with respect to *X* if all $x \in X$ can access *I*
- Types of availability:
  - Traditional: *x* gets access or not
  - Quality of service: promised a level of access (for example, a specific level of bandwidth); *x* meets it or not, even though some access is achieved

Policy Models

- Abstract description of a policy or class of policies
- Focus on points of interest in policies
  - Security levels in multilevel security models
  - Separation of duty in Clark-Wilson model
  - Conflict of interest in Chinese Wall model

Mechanisms

- Entity or procedure that enforces some part of the security policy
  - Access controls (like bits to prevent someone from reading a homework file)
  - Disallowing people from bringing CDs and floppy disks into a computer facility to control what is placed on systems

Types of Security Policies

- Military (governmental) security policy
  - Policy primarily protecting confidentiality
- Commercial security policy
  - Policy primarily protecting integrity
- Confidentiality policy
  - Policy protecting only confidentiality
- Integrity policy
  - Policy protecting only integrity

Integrity and Transactions

- Begin in consistent state
  - “Consistent” defined by specification
- Perform series of actions (*transaction*)
  - Actions cannot be interrupted
  - If actions complete, system in consistent state
  - If actions do not complete, system reverts to a consistent state

Types of Access Control

- Discretionary Access Control (DAC, IBAC)
  - Individual user sets access control mechanism to allow or deny access to an object
- Mandatory Access Control (MAC)
  - System mechanism controls access to object, and individual cannot alter that access
- Originator Controlled Access Control (ORCON, ORGCON)
  - Originator (creator) of information controls who can access information