ECS 235B, Lecture 4

January 14, 2019

Safety Question

- In this model:

  Is it possible to have a derivable state with **X**/*r*:*c* in *dom*(**A**), or does there exist a subject **B** with ticket **X**/*rc* in the initial state or which can demand **X**/*rc* and τ(**X**)/*r*:*c* in *flow**(**B**,**A**)?

- To answer: construct maximal state and test
  - Consider acyclic attenuating schemes; how do we construct maximal state?

Intuition

- Consider state *h*.
- State *u* corresponds to *h* but with minimal number of new entities created such that maximal state *m* can be derived with no create operations
  - So if in history from *h* to *m*, subject **X** creates two entities of type *a*, in *u* only one would be created; surrogate for both
- *m* can be derived from *u* in polynomial time, so if *u* can be created by adding a finite number of subjects to *h*, safety question decidable.

Fully Unfolded State

- State *u* derived from state 0 as follows:
  - delete all loops in *cc*; new relation *cc′*
  - mark all subjects as folded
  - while any **X** $\in$ SUB^0 is folded
    - mark it unfolded
    - if **X** can create entity **Y** of type *y*, it does so (call this the *y*-surrogate of **X**); if entity **Y** $\in$ SUB^g, mark it folded
  - if any subject in state *h* can create an entity of its own type, do so
- Now in state *u*

Termination

- First loop terminates as SUB^0 finite
- Second loop terminates:
  - Each subject in SUB^0 can create at most $|\ TS\ |$ children, and $|\ TS\ |$ is finite
  - Each folded subject in $|\ SUB^i\ |$ can create at most $|\ TS\ | - i$ children
  - When $i = |\ TS\ |$, subject cannot create more children; thus, folded is finite
  - Each loop removes one element
- Third loop terminates as SUB^h is finite

Surrogate

- Intuition: surrogate collapses multiple subjects of same type into single subject that acts for all of them
- Definition: given initial state 0, for every derivable state *h* define *surrogate function* σ:$ENT^h \rightarrow ENT^h$ by:
  - if **X** in ENT^0, then σ(**X**) = **X**
  - if **Y** creates **X** and τ(**Y**) = τ(**X**), then σ(**X**) = σ(**Y**)
  - if **Y** creates **X** and τ(**Y**) ≠ τ(**X**), then σ(**X**) = τ(**Y**)-surrogate of σ(**Y**)

Implications

- $\tau(\sigma(\mathbf{X})) = \tau(\mathbf{X})$
- If $\tau(\mathbf{X}) = \tau(\mathbf{Y})$, then $\sigma(\mathbf{X}) = \sigma(\mathbf{Y})$
- If $\tau(\mathbf{X}) \neq \tau(\mathbf{Y})$, then
  - $\sigma(\mathbf{X})$ creates $\sigma(\mathbf{Y})$ in the construction of u
  - $\sigma(\mathbf{X})$ creates entities $\mathbf{X}'$ of type $\tau(\mathbf{X}') = \tau(\sigma(\mathbf{X}))$
- From these, for a system with an acyclic attenuating scheme, if **X** creates **Y**, then tickets that would be introduced by pretending that $\sigma(\mathbf{X})$ creates $\sigma(\mathbf{Y})$ are in $dom^u(\sigma(\mathbf{X}))$ and $dom^u(\sigma(\mathbf{Y}))$

Deriving Maximal State

- Idea
  - Reorder operations so that all creates come first and replace history with equivalent one using surrogates
  - Show maximal state of new history is also that of original history
  - Show maximal state can be derived from initial state

Reordering

- *H* legal history deriving state *h* from state 0
- Order operations: first create, then demand, then copy operations
- Build new history *G* from *H* as follows:
  - Delete all creates
  - "**X** demands **Y**/*r*:*c*" becomes "σ(**X**) demands σ(**Y**)/*r*:*c*"
  - "**Y** copies **X** /*r*:*c* from **Y**" becomes "σ(**Y**) copies σ(**X**)/*r*:*c* from σ(**Y**)"

Tickets in Parallel

- Lemma
  - All transitions in G legal; if $\mathbf{X}/r{:}c \in dom^h(Y)$, then $\sigma(\mathbf{X})/r{:}c \in dom^h(\sigma(\mathbf{Y}))$
- Outline of proof: induct on number of copy operations in H

Basis

- *H* has create, demand only; so *G* has demand only. s preserves type, so by construction every demand operation in *G* legal.
- 3 ways for **X**/*r*:*c* to be in dom^h(**Y**):
  - **X**/*r*:*c* $\in$ dom^0(**Y**) means **X**, **Y** $\in$ ENT^0, so trivially σ(**X**)/*r*:*c* $\in$ dom^g(σ(**Y**)) holds
  - A create added **X**/*r*:*c* $\in$ dom^h(**Y**): previous lemma says σ(**X**)/*r*:*c* $\in$ dom^g(σ(**Y**)) holds
  - A demand added **X**/*r*:*c* $\in$ dom^h(**Y**): corresponding demand operation in *G* gives σ(**X**)/*r*:*c* $\in$ dom^g(σ(**Y**))

Hypothesis

- Claim holds for all histories with k copy operations
- History H has k+1 copy operations
  - H' initial sequence of H composed of k copy operations
  - h' state derived from H'

Step

- G' sequence of modified operations corresponding to H'; g' derived state
  - G' legal history by hypothesis
- Final operation is "Z copied X/r:c from Y"
  - So h, h' differ by at most $\mathbf{X}/r{:}c \in dom^h(\mathrm{Z})$
  - Construction of G means final operation is $\sigma(\mathbf{X})/r{:}c \in dom^g(\sigma(\mathbf{Y}))$
- Proves second part of claim

Step

- H' legal, so for H to be legal, we have:
  1. $\mathbf{X}/rc \in dom^{h'}(\mathbf{Y})$
  2. $link_i^{h'}(\mathbf{Y}, \mathbf{Z})$
  3. $\tau(\mathbf{X}/r{:}c) \in f_i(\tau(\mathbf{Y}), \tau(\mathbf{Z}))$
- By IH, 1, 2, as $\mathbf{X}/\mathrm{r{:}c} \in dom^{h'}(\mathbf{Y})$,

$$\sigma(\mathbf{X})/r{:}c \in dom^{g'}(\sigma(\mathbf{Y})) \text{ and } link_i^{g'}(\sigma(\mathbf{Y}), \sigma(\mathbf{Z}))$$

- As σ preserves type, IH and 3 imply

$$\tau(\sigma(\mathbf{X})/r{:}c) \in f_i(\tau((\sigma(\mathbf{Y})), \tau(\sigma(\mathbf{Z})))$$

- IH says G' legal, so G is legal

Corollary

- If $link_i^h(\mathbf{X}, \mathbf{Y})$, then $link_i^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y}))$

Main Theorem

- System has acyclic attenuating scheme
- For every history *H* deriving state *h* from initial state, there is a history *G* without create operations that derives *g* from the fully unfolded state *u* such that

$$(\forall \mathbf{X},\mathbf{Y} \in SUB^h)[flow^h(\mathbf{X}, \mathbf{Y}) \subseteq flow^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y}))]$$

- Meaning: any history derived from an initial state can be simulated by corresponding history applied to the fully unfolded state derived from the initial state

Proof

- Outline of proof: show that every $path^h(\mathbf{X},\mathbf{Y})$ has corresponding $path^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y}))$ such that $cap(path^h(\mathbf{X},\mathbf{Y})) = cap(path^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y})))$
  - Then corresponding sets of tickets flow through systems derived from *H* and *G*
  - As initial states correspond, so do those systems
- Proof by induction on number of links

Basis and Hypothesis

- Length of $path^h(\mathbf{X}, \mathbf{Y}) = 1$. By definition of $path^h$, $link_i^h(\mathbf{X}, \mathbf{Y})$, hence $link_i^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y}))$. As σ preserves type, this means

$$cap(path^h(\mathbf{X}, \mathbf{Y})) = cap(path^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y})))$$

- Now assume this is true when $path^h(\mathbf{X}, \mathbf{Y})$ has length k

Step

- Let $path^h(\mathbf{X}, \mathbf{Y})$ have length k+1. Then there is a **Z** such that $path^h(\mathbf{X}, \mathbf{Z})$ has length k and $link_j^h(\mathbf{Z}, \mathbf{Y})$.
- By IH, there is a $path^g(\sigma(\mathbf{X}), \sigma(\mathbf{Z}))$ with same capacity as $path^h(\mathbf{X}, \mathbf{Z})$
- By corollary, $link_j^g(\sigma(\mathbf{Z}), \sigma(\mathbf{Y}))$
- As σ preserves type, there is $path^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y}))$ with

$$cap(path^h(\mathbf{X}, \mathbf{Y})) = cap(path^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y})))$$

Implication

- Let maximal state corresponding to *v* be *#u*
  - Deriving history has no creates
  - By theorem,

$$(\forall \mathbf{X},\mathbf{Y} \in SUB^h)[flow^h(\mathbf{X}, \mathbf{Y}) \subseteq flow^{\#u}(\sigma(\mathbf{X}), \sigma(\mathbf{Y}))]$$

  - If $\mathbf{X} \in SUB^0$, $\sigma(\mathbf{X}) = \mathbf{X}$, so:

$$(\forall \mathbf{X},\mathbf{Y} \in SUB^0)[flow^h(\mathbf{X}, \mathbf{Y}) \subseteq flow^{\#u}(\mathbf{X}, \mathbf{Y})]$$

- So *#u* is maximal state for system with acyclic attenuating scheme
  - *#u* derivable from *u* in time polynomial to $|SUB^u|$
  - Worst case computation for $flow^{\#u}$ is exponential in $|TS|$

Safety Result

- If the scheme is acyclic and attenuating, the safety question is decidable

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
  - SMP has no equivalent to delete, destroy
- HRU allows multiparent creates
  - SMP 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 r in a[s0, s1] and r 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

- *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
  - $cr_{\mathrm{P},i}(\tau(\mathbf{X}_1), \ldots, \tau(\mathbf{X}_n)) = \mathbf{Y}/R_{1,1} \cup \mathbf{X}_i/R_{2,i}$
  - $cr_{\mathrm{C}}(\tau(\mathbf{X}_1), \ldots, \tau(\mathbf{X}_n)) = \mathbf{Y}/R_3 \cup \mathbf{X}_1/R_{4,1} \cup \ldots \cup \mathbf{X}_n/R_{4,n}$

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 $\mathbf{C}$):
  - $cc(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = Z \subseteq T$
  - $cr_{\mathbf{P}1}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{1,1} \cup \mathbf{P}_1/R_{2,1}$
  - $cr_{\mathbf{P}2}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{2,1} \cup \mathbf{P}_2/R_{2,2}$
  - $cr_{\mathbf{P}3}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{3,1} \cup \mathbf{P}_3/R_{2,3}$

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 $\mathbf{X}/t \in dom(\mathbf{Y})$, **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 $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, $\mathbf{S}$, $\mathbf{C}$; 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 $\mathbf{P}_3/Rtc \cup \mathbf{A}_2/tc$
- $\mathbf{S}$ has $\mathbf{A}_3/tc \cup \mathbf{C}/Rtc$
- $\mathbf{C}$ 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 $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, **C**
- If create of **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

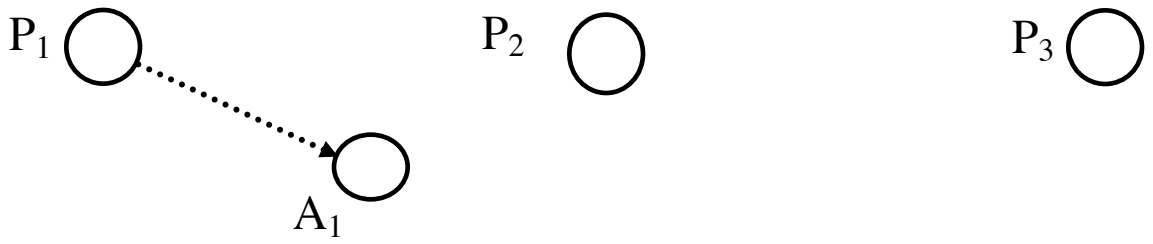
- Monotonic ESPM and the monotonic HRU model are equivalent.
- Safety question in ESPM also decidable if acyclic attenuating scheme
  - Proof similar to that for SPM

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

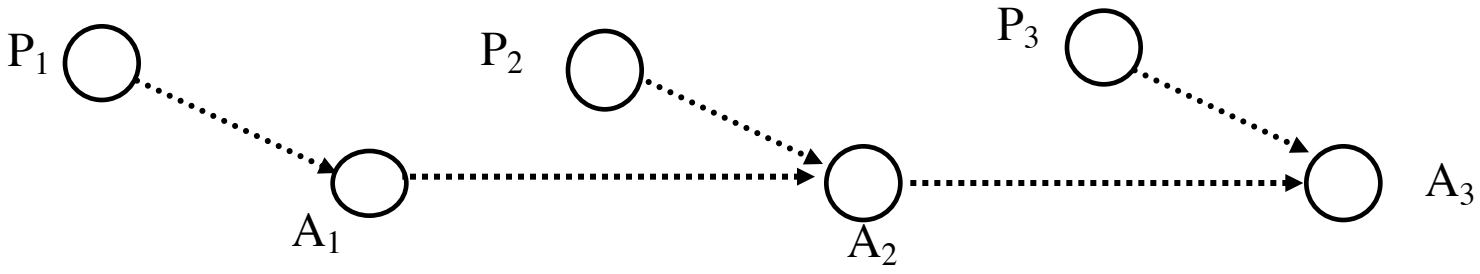
Example: 3-Parent Joint Creation

- 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´*



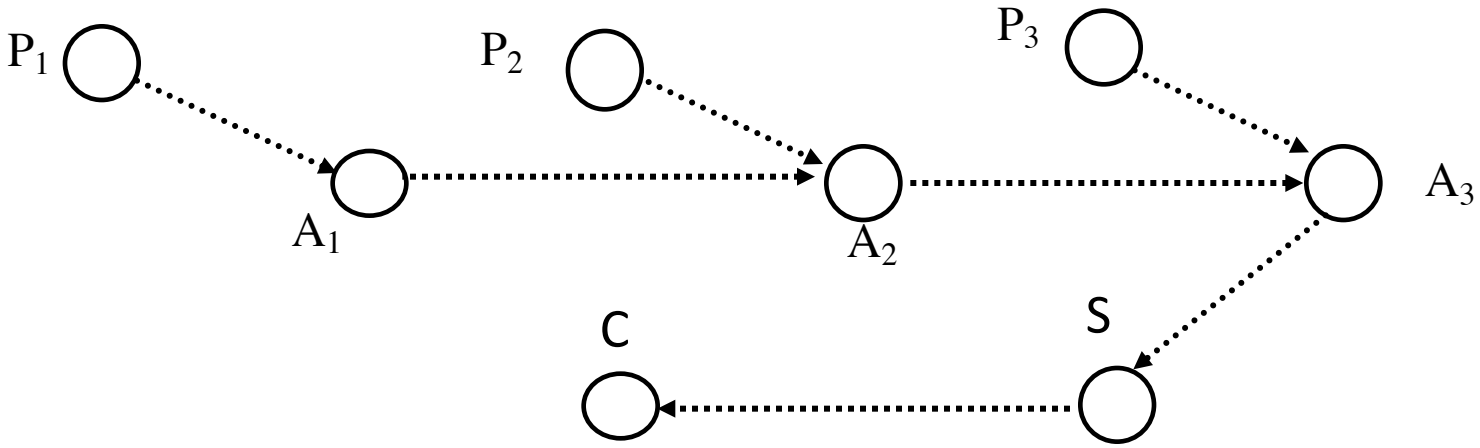
Next Step

- **A_1**, **P_2** create **A_2**; **A_2**, **P_3** create **A_3**
- Type of nodes, edges are *a* and *e´*



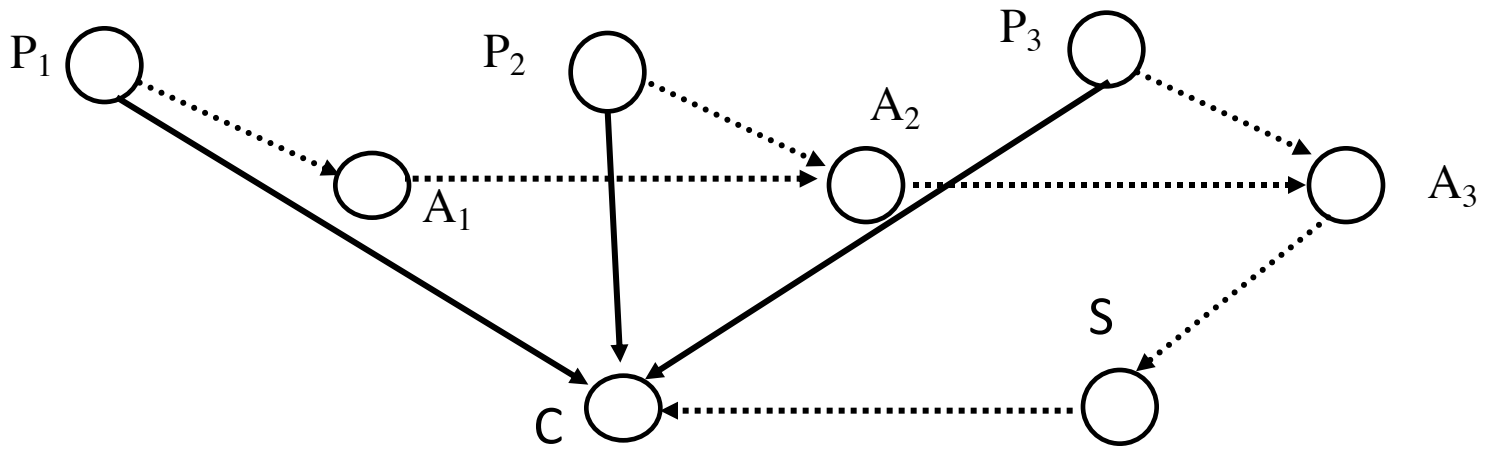
Next Step

- **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

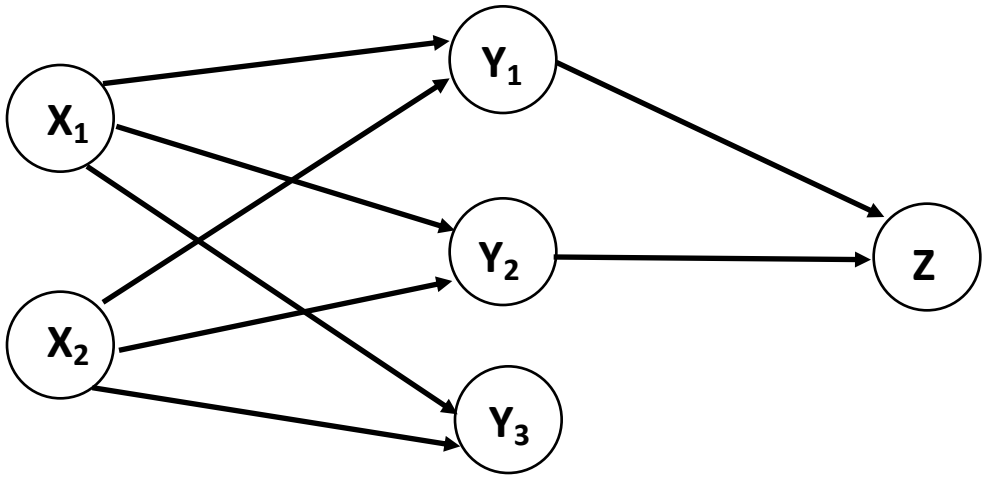
- All nodes in *A* have even number of incoming edges
  - 2-parent create adds 2 incoming edges
- Edge adding operation in *B* that can edge from $\mathbf{X}_2$ to **C** can add one from $\mathbf{X}_3$ to **C**
  - *A* cannot enter this state
  - *B* cannot transition to a state in which **Y** has even 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

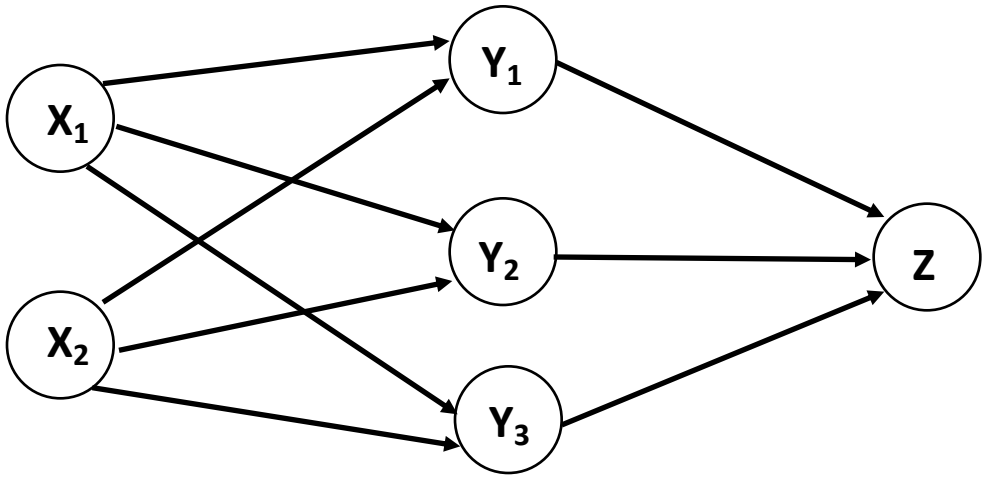
Outline of Proof

- $\mathbf{X}_1$, $\mathbf{X}_2$ 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)
  - $\tau: O \rightarrow T$ specifies type of each object
  - If **X** subject, $\tau(\mathbf{X})$ in TS
  - If **X** object, $\tau(\mathbf{X})$ 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:
  - $S' = S, O' = O \cup \{ o \}$
  - $(\forall y \in O)[\tau'(y) = \tau(y)], \tau'(o) = t$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

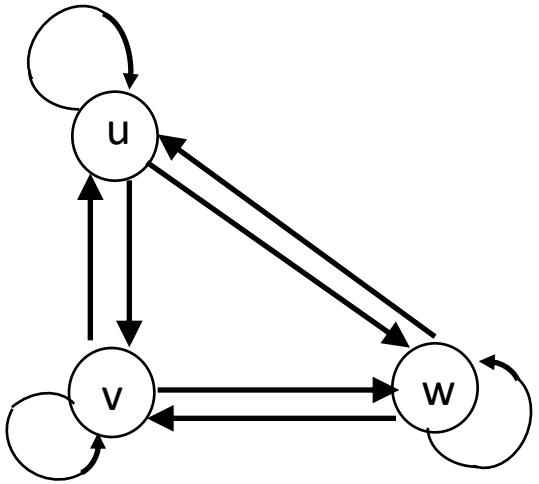
Definitions

- 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(s1 : u, s2 : u, o1 : v, o2 : v,
                                  o3 : w, o4 : w)
  create subject s1 of type u;
  create object o1 of type v;
  create object o3 of type w;
  enter r into a[s2, s1];
  enter r into a[s2, o2];
  enter r into a[s2, o4]
end
```

Creation Graph

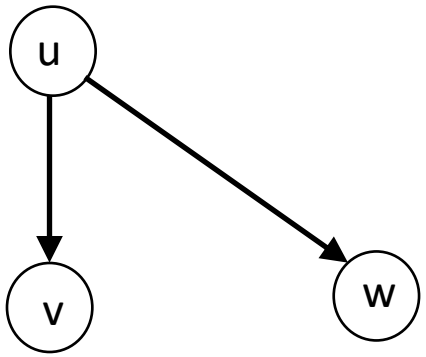


- *u*, *v*, *w* child types
- *u*, *v*, *w* also parent types
- Graph: lines from parent types to child types
- This one has cycles

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

Alternate Definition of "Scheme"

- Σ set of states
- Q set of queries
- $e: \Sigma \times Q \rightarrow \{true, false\}$
  - 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 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 $\forall$ or $\exists$
- If Π is $\exists$, *existential* security analysis
  - Is there a state σ' such that $\sigma \mapsto_\tau^* \sigma'$, $e(\sigma', q) = true$?
- If Π is $\forall$, *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