Islands

- *tg*-path: path of distinct vertices connected by edges labeled *t* or *g*
  - Call them "tg-connected"
- island: maximal *tg*-connected subject-only subgraph
  - Any right one vertex has can be shared with any other vertex

Initial, Terminal Spans

- *initial span* from **x** to **y**
  - **x** subject
  - *tg*-path between **x**, **y** with word in { $\overrightarrow{t^*}\overrightarrow{g}$ } ∪ { ν }
  - Means **x** can give rights it has to **y**
- *terminal span* from **x** to **y**
  - **x** subject
  - *tg*-path between **x**, **y** with word in { $\overrightarrow{t^*}$ } ∪ { ν }
  - Means **x** can acquire any rights **y** has

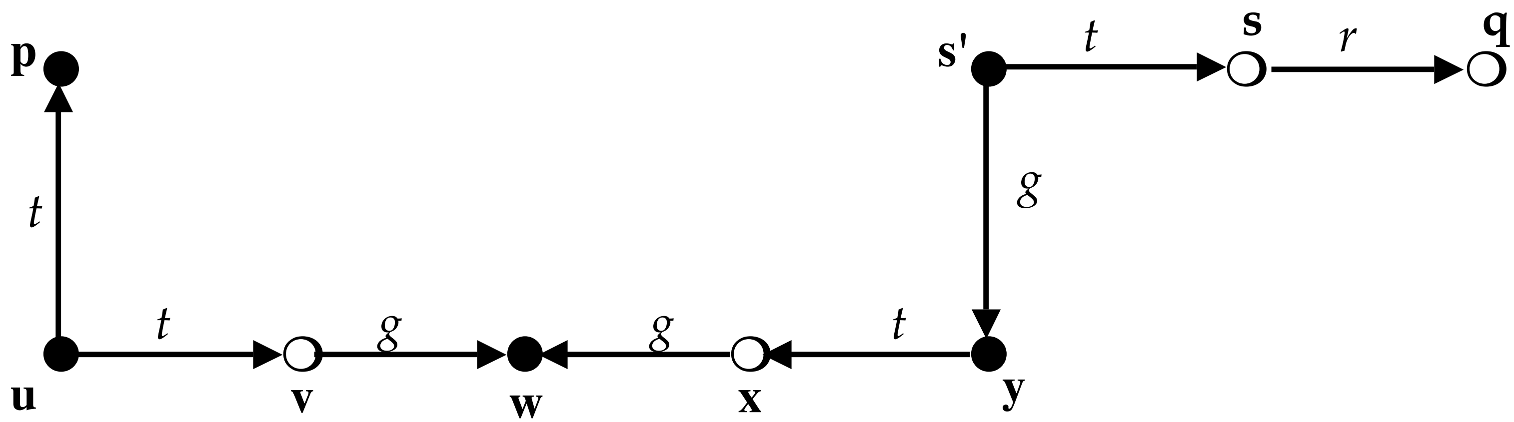
Bridges

- bridge: *tg*-path between subjects **x**, **y**, with associated word in

$$\{\ \overrightarrow{t}^{*}, \overrightarrow{t}^{*}, \overrightarrow{t}^{*}\overleftarrow{g}\ \overleftarrow{t}^{*}, \overrightarrow{t}^{*}\overrightarrow{g}\ \overleftarrow{t}^{*}\ \}$$

  - rights can be transferred between the two endpoints
  - *not* an island as intermediate vertices are objects

Example



- islands { p, u } { w } { y, s' }
- bridges u, v, w; w, x, y
- initial span p (associated word ν)
- terminal span s's (associated word $\vec{t}$)

can•share Predicate

Definition:

- *can•share*(r, **x**, **y**, G_0) if, and only if, there is a sequence of protection graphs G_0, …, G_n such that $G_0 \vdash^* G_n$ using only *de jure* rules and in G_n there is an edge from **x** to **y** labeled r.

can•share Theorem

- *can•share*(r, **x**, **y**, G_0) if, and only if, there is an edge from **x** to **y** labeled r in G_0, or the following hold simultaneously:
  - There is an **s** in G_0 with an **s**-to-**y** edge labeled r
  - There is a subject **x′** = **x** or initially spans to **x**
  - There is a subject **s′** = **s** or terminally spans to **s**
  - There are islands $I_1, \ldots, I_k$ connected by bridges, and **x′** in I_1 and **s′** in I_k

Outline of Proof

- **s** has *r* rights over **y**
- **s**′ acquires *r* rights over **y** from **s**
  - Definition of terminal span
- **x**′ acquires *r* rights over **y** from **s**′
  - Repeated application of sharing among vertices in islands, passing rights along bridges
- **x**′ gives *r* rights over **y** to **x**
  - Definition of initial span

Key Question

- Characterize class of models for which safety is decidable
  - Existence: Take-Grant Protection Model is a member of such a class
  - Universality: In general, question undecidable, so for some models it is not decidable
- What is the dividing line?

Schematic Protection Model

- Type-based model
  - Protection type: entity label determining how control rights affect the entity
    - Set at creation and cannot be changed
  - Ticket: description of a single right over an entity
    - Entity has sets of tickets (called a *domain*)
    - Ticket is **X**/*r*, where **X** is entity and *r* right
  - Functions determine rights transfer
    - Link: are source, target "connected"?
    - Filter: is transfer of ticket authorized?

Link Predicate

- Idea: $link_i(\mathbf{X}, \mathbf{Y})$ if **X** can assert some control right over **Y**
- Conjunction of disjunction of:
  - $\mathbf{X}/z \in dom(\mathbf{X})$
  - $\mathbf{X}/z \in dom(\mathbf{Y})$
  - $\mathbf{Y}/z \in dom(\mathbf{X})$
  - $\mathbf{Y}/z \in dom(\mathbf{Y})$
  - **true**

Examples

- Take-Grant:

  $\mathit{link}(\mathbf{X}, \mathbf{Y}) = \mathbf{Y}/g \in \mathit{dom}(\mathbf{X}) \vee \mathbf{X}/t \in \mathit{dom}(\mathbf{Y})$

- Broadcast:

  $\mathit{link}(\mathbf{X}, \mathbf{Y}) = \mathbf{X}/b \in \mathit{dom}(\mathbf{X})$

- Pull:

  $\mathit{link}(\mathbf{X}, \mathbf{Y}) = \mathbf{Y}/p \in \mathit{dom}(\mathbf{Y})$

Filter Function

- Range is set of copyable tickets
  - Entity type, right
- Domain is subject pairs
- Copy a ticket $\mathbf{X}/r{:}c$ from $dom(\mathbf{Y})$ to $dom(\mathbf{Z})$
  - $\mathbf{X}/rc \in dom(\mathbf{Y})$
  - $link_i(\mathbf{Y}, \mathbf{Z})$
  - $\tau(\mathbf{Y})/r{:}c \in f_i(\tau(\mathbf{Y}), \tau(\mathbf{Z}))$
- One filter function per link function

Example

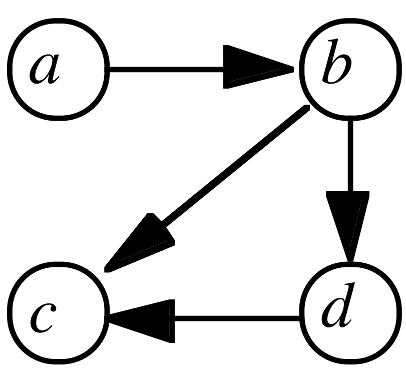
- $f(\tau(\mathbf{Y}), \tau(\mathbf{Z})) = T \times R$
  – Any ticket can be transferred (if other conditions met)
- $f(\tau(\mathbf{Y}), \tau(\mathbf{Z})) = T \times RI$
  – Only tickets with inert rights can be transferred (if other conditions met)
- $f(\tau(\mathbf{Y}), \tau(\mathbf{Z})) = \varnothing$
  – No tickets can be transferred

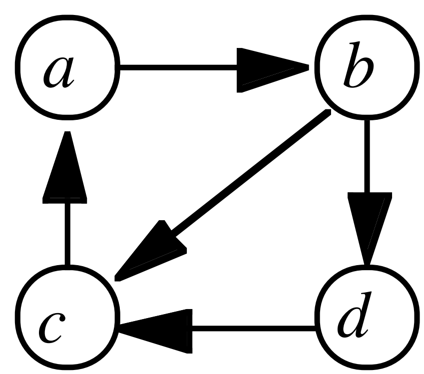
Example

- Take-Grant Protection Model
  - TS = { subjects }, TO = { objects }
  - $RC = \{ tc, gc \}$, $RI = \{ rc, wc \}$
  - $link(\mathbf{p}, \mathbf{q}) = \mathbf{p}/t \in dom(\mathbf{q}) \vee \mathbf{q}/t \in dom(\mathbf{p})$
  - $f(subject, subject) = \{ subject, object \} \times \{ tc, gc, rc, wc \}$

Create Operation

- Must handle type, tickets of new entity
- Relation *can•create*(*a*, *b*)
  - Subject of type *a* can create entity of type *b*
- Rule of acyclic creates:





Types

- $cr(a, b)$: tickets introduced when subject of type a creates entity of type b
- **B** object: $cr(a, b) \subseteq \{ b/r{:}c \in RI \}$
- **B** subject: $cr(a, b)$ has two parts
  - $cr_P(a, b)$ added to **A**, $cr_C(a, b)$ added to **B**
  - **A** gets **B**/$r{:}c$ if $b/r{:}c$ in $cr_P(a, b)$
  - **B** gets **A**/$r{:}c$ if $a/r{:}c$ in $cr_C(a, b)$

Non-Distinct Types

$cr(a, a)$: who gets what?

- $self/r{:}c$ are tickets for creator
- $a/r{:}c$ tickets for created

$cr(a, a) = \{\ a/r{:}c,\ self/r{:}c \mid r{:}c \in R\}$

Attenuating Create Rule

$cr(a, b)$ attenuating if:

1. $cr_C(a, b) \subseteq cr_P(a, b)$ and
2. $a/r{:}c \in cr_P(a, b) \Rightarrow \mathit{self}/r{:}c \in cr_P(a, b)$

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

- can•create 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 $x \in R$
  - $cc(a, a) = p$
  - $cr_{\text{Anna}}(a, a, p) = cr_{\text{Bill}}(a, a, p) = \varnothing$
  - $cr_{\text{proxy}}(a, a, p) = \{$ Anna/x, Bilł/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
  - $link_1(\mathbf{A}_1, \mathbf{A}_2) = \mathbf{A}_1/t \in dom(\mathbf{A}_2) \wedge \mathbf{A}_2/t \in dom(\mathbf{A}_2)$
  - $link_1(\mathbf{A}_2, \mathbf{A}_3) = \mathbf{A}_2/t \in dom(\mathbf{A}_3) \wedge \mathbf{A}_3/t \in dom(\mathbf{A}_3)$
  - $link_2(\mathbf{S}, \mathbf{A}_3) = \mathbf{A}_3/t \in dom(\mathbf{S}) \wedge \mathbf{C}/t \in dom(\mathbf{C})$
  - $link_3(\mathbf{A}_1, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_1)$
  - $link_3(\mathbf{A}_2, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_2)$
  - $link_3(\mathbf{A}_3, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_3)$
  - $link_4(\mathbf{A}_1, \mathbf{P}_1) = \mathbf{P}_1/t \in dom(\mathbf{A}_1) \wedge \mathbf{A}_1/t \in dom(\mathbf{A}_1)$
  - $link_4(\mathbf{A}_2, \mathbf{P}_2) = \mathbf{P}_2/t \in dom(\mathbf{A}_2) \wedge \mathbf{A}_2/t \in dom(\mathbf{A}_2)$
  - $link_4(\mathbf{A}_3, \mathbf{P}_3) = \mathbf{P}_3/t \in dom(\mathbf{A}_3) \wedge \mathbf{A}_3/t \in dom(\mathbf{A}_3)$

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 $\mathbf{C}/R_3$

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 $\mathbf{P}_2/Rtc \cup \mathbf{A}_1/tc \cup \mathbf{A}_1/t \cup \mathbf{C}/Rtc$
- 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 $link_4$s true ⇒ 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

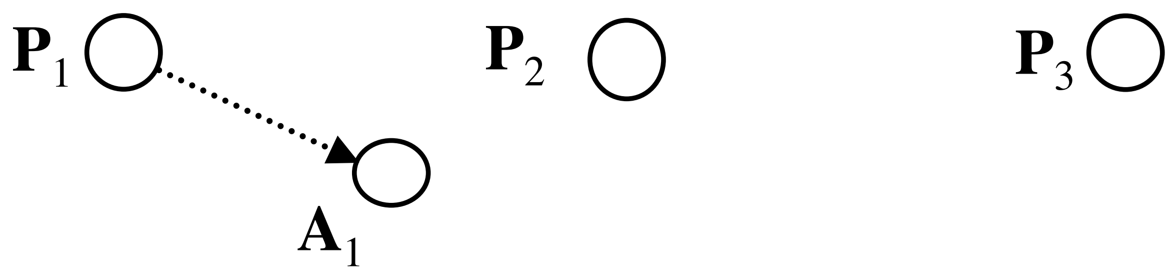
- Monotonic ESPM and the monotonic HRU model are equivalent.
- Safety question in ESPM also decidable if acyclic attenuating scheme

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

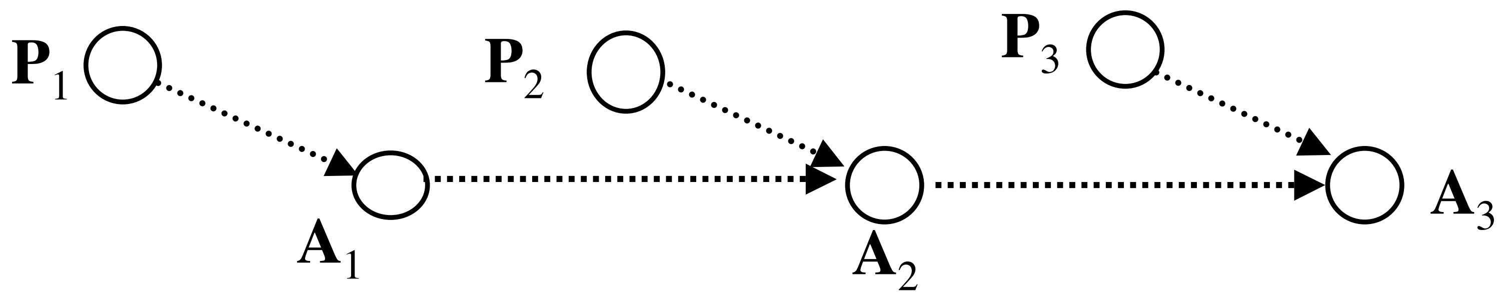
Example: 3-Parent Joint Creation

- 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´*



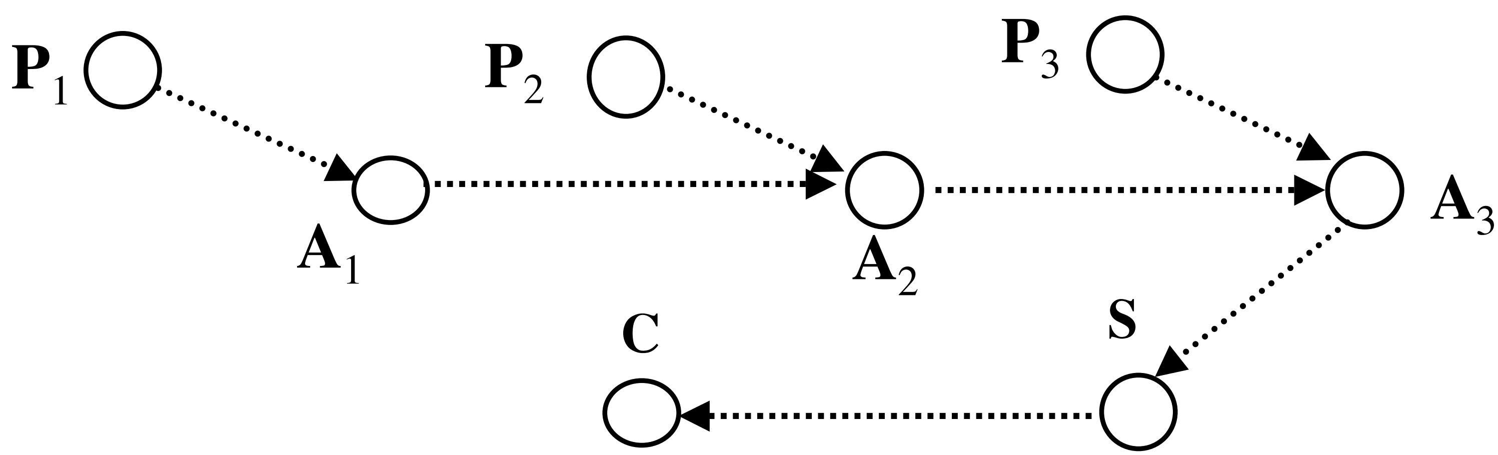
Next Step

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



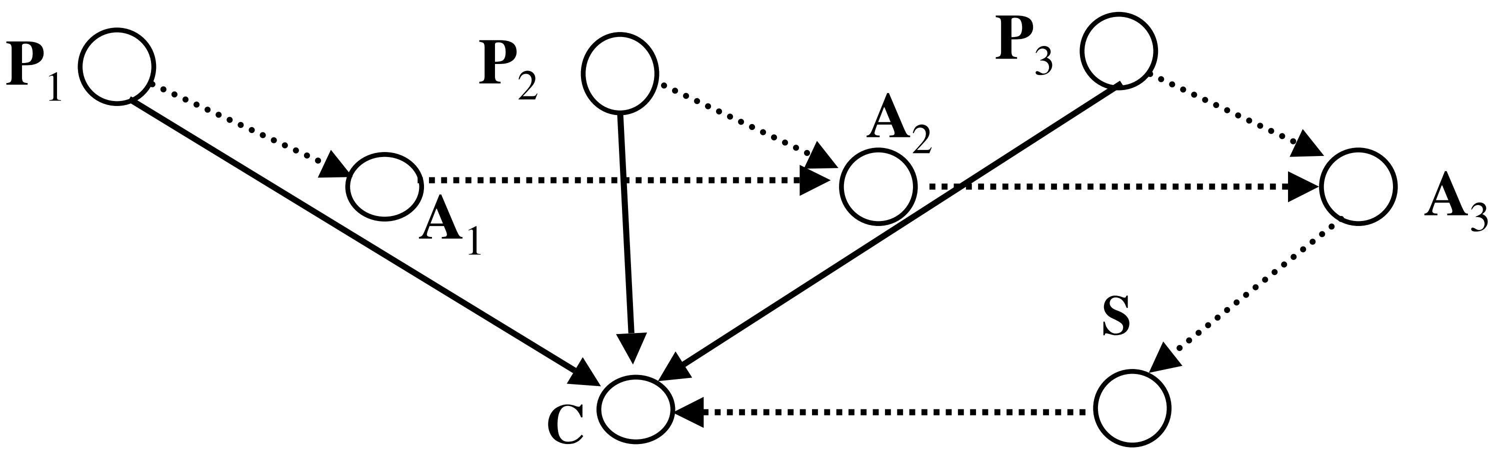
Next Step

- $\mathbf{A}_3$ 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 scheme in *MA* 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
- 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, $\tau(\mathbf{X})$ 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
- $\alpha(x_1{:}t_1, ..., x_n{:}t_n)$ 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 *havoc*($s_1 : u, s_2 : u, o_1 : v, o_2 : v, o_3 : w, o_4 : \mathrm{w}$)

create subject s_1 **of type** u;

create object o_1 **of type** v;

create object o_3 **of type** w;

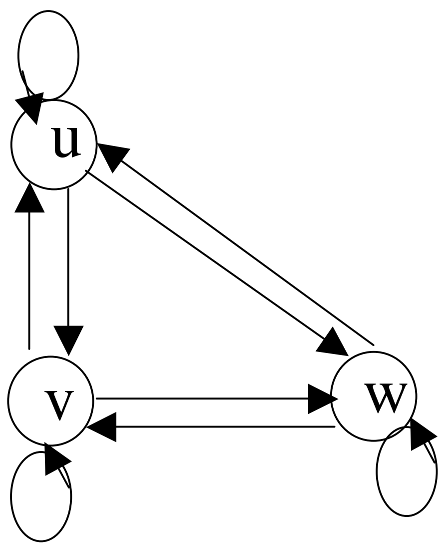
enter r **into** $a[s_2, s_1]$;

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

Theorems

- Safety decidable for systems with acyclic MTAM schemes
- 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

Key Points

- Safety problem undecidable
- Limiting scope of systems can make problem decidable
- Types critical to safety problem's analysis