Take-Grant Protection Model

- A specific (not generic) system
  - Set of rules for state transitions
- Safety decidable, and in time linear with the size of the system
- Goal: find conditions under which rights can be transferred from one entity to another in the system

Jones, Lipton, Snyder FOCS’76
System

- objects (files, …)
- subjects (users, processes, …)
- don’t care (either a subject or an object)

G \xrightarrow{x} G'  
apply a rewriting rule x (witness) to G to get G'

G \xrightarrow{*} G'  
apply a sequence of rewriting rules (witness) to G to get G'

R = \{ t, g, r, w, … \}  
set of rights

Take-Grant Protection Model

- System is directed graph
  - Subject: ●
  - Object: ○
  - (labeled) edge: \{rights\}
- Take rule: if \( t \in \gamma, \alpha \subseteq \beta \), can add transitive edge
- Grant rule: if \( g \in \zeta, \alpha \subseteq \gamma \), can add (grant) edge between recipients
- Create, Remove rules
Take-Grant Protection Model: Sharing

- Given $G_0$, can vertex $x$ obtain $\alpha$ rights over $y$?
  - $\text{Can}_\text{share}(\alpha, x, y, G_0)$ iff $G_0 \vdash^* G_n$ using the above rules and $\alpha$ edge from $x$ to $y$ in $G_n$

- $tg$-$path$: $v_0, \ldots, v_n$ where $t$ or $g$ edge between any $v_i, v_{i+1}$
  - Vertices $tg$-$connected$ if $tg$-$path$ between them

- Theorem: Any two subjects with $tg$-$path$ of length 1 can cause rights to be shared

Any two subjects with $tg$-$path$ of length 1 can share rights

$\text{Can}_\text{share}(\alpha, x, y, G_0)$

- Four possible length 1 $tg$-paths
- Take rule
- Grant rule
- Sequence:
  - Create
  - Take
  - Grant
  - Take
Other definitions

- Island: Maximal $tg$-connected subject-only subgraph
  - Can_share all rights in island
  - Proof: Induction from previous theorem

- Bridge: $tg$-path between subjects $v_0$ and $v_n$ with edges of the following form:
  - All $t$
  - $0+ t$ increasing, $g$, $0+ t$ decreasing

Example

- islands: \{ p, u \} \{ w \} \{ y, s' \}
- bridges: u, v, w; w, x, y
- initial span: p (associated word v)
- terminal span: s's (associated word t)
Theorem: Can_share(\(\alpha,x,y,G_0\)) (for subjects)

- Can_share(\(\alpha,x,y,G_0\)) if \(x\) and \(y\) are subjects and there is an \(\alpha\) edge from \(x\) to \(y\) in \(G_0\) or if:
  - \(\exists\) a subject \(s \in G_0\) with an \(s\) to \(y\) \(\alpha\) edge, and
  - \(\exists\) islands \(I_1, \ldots, I_n\) such that \(x \in I_1, s \in I_n\), and there is a bridge from \(I_j\) to \(I_{j+1}\)

- Proof: Islands above, bridge – take in both directions to grant link, then one takes “grant” and grants to other

- If \(x\) and \(y\) are subjects, “only if” holds
  - If no take/grant or two grants between objects, can’t bridge gap. Otherwise it is either a bridge or an island

What about objects?

- \(x\) *initially spans* to \(y\) if \(x\) is a subject and there is a \(tg\)-path between them with \(t\) edges ending in a \(g\) edge
  - \(x\) can grant a right to \(y\)

- \(x\) *terminally spans* to \(y\) if \(x\) is a subject and there is a \(tg\)-path between them with \(t\) edges
  - \(x\) can take a right from \(y\)
Theorem: Can_share(α,x,y,G₀)

• Can_share(α,x,y,G₀) iff there is an α edge from x to y in G₀ or if:
  – ∃ a vertex s ∈ G₀ with an s to y α edge,
  – ∃ a subject x' such that x'=x or x' initially spans to x,
  – ∃ a subject s' such that s'=s or s' terminally spans to s, and
  – ∃ islands I₁, ..., Iₙ such that x' ∈ I₁, s' ∈ Iₙ, and there is a bridge from Iᵢ to Iᵢ₊₁

• Proof: If: x' grants to x, s’ takes from s, otherwise as with subjects
  – Only if: as before, plus object can't give (receive) a right unless someone can take (grant) it

• Corollary: There is an O(|V|+|E|) algorithm to test can_share
Creating models from scratch

- $G_0 = \bullet, R$ a set of rights. $G_0 \vdash^* G$ iff $G$ is a finite directed acyclic graph, edges labeled from $R$, and at least one subject with no incoming edge.
  - If: construction (create)
  - Only if: Can’t add an edge to initial subject

- A $k$-component, $n$-edge protection graph can be constructed from $t$-rule applications, where $2(k-1)+n \leq t \leq 2(k-1)+3n$

Use of the model

- Sharing rights with trusted entity
- Stealing (rights available with non-cooperating subjects)
- Collusion
Sharing Rights through Trusted Entity

• Subjects $p$ and $q$ communicate through buffer object $b$
  – Trusted entity $s$ controls access to $b$
  – $p$ and $q$ have private information $u$ and $v$

Theft

• Can_steal($\alpha, x, y, G_0$) if there is no $\alpha$ edge from $x$ to $y$ in $G_0$ and $\exists G_1, \ldots, G_n$ s. t.:
  – $\exists \alpha$ edge from $x$ to $y$ in $G_n$,
  – $\exists$ rules $\rho_1, \ldots, \rho_n$ that take $G_{i-1} \vdash G_i$, and
  – $\forall v, w \in G_i, 1 \leq i \leq n$, if $\exists \alpha$ edge from $v$ to $y$ in $G_0$ then $\rho_i$ is not “$v$ grants ($\alpha$ to $y$) to $w$”
• Ideal: Steal possible if $x$ gets $\alpha$ on $y$
  without anyone granting $\alpha$ on $y$ to anyone
Theorem: When Theft Possible

- Can_steal(\(\alpha, x, y, G_0\)) iff there is no \(\alpha\) edge from \(x\) to \(y\) in \(G_0\) and \(\exists G_1, \ldots, G_n\) s. t.:
  - \(\exists\) subject \(x'\) such that \(x'=x\) or \(x'\) initially spans to \(x\), and
  - \(\exists s\) with \(\alpha\) edge to \(y\) in \(G_0\) and can_share(\(t, x', s, G_0\))

- Proof:
  - \(\Rightarrow\): (easy – build path)
  - \(\Leftarrow\): Assume can_steal:
    - No \(\alpha\) edge from definition.
    - Can_share(\(\alpha, x, y, G_0\)) from definition: \(\alpha\) from \(x\) to \(y\) in \(G_n\)
    - \(s\) exists from can_share and Monday’s theorem
    - Can_share(\(t, x', s, G_0\)): \(s\) can’t grant \(\alpha\) (definition), someone else must get \(\alpha\) from \(s\), show that this can only be accomplished with take rule

Conspiracy

How many subjects needed to enable Can_share(\(\alpha, x, y, G_0\))?

- Access set \(A(y)\) with focus \(y\) is set of vertices \(y\) \(\cup\) vertices to which \(y\) initially spans \(\cup\) vertices to which \(y\) terminally spans
- Deletion set \(\delta(y, y')\): All \(z \in A(y) \cap A(y')\) for which
  - \(y\) initially spans to \(z\) and \(y'\) terminally spans to \(z\) \(\cup\)
  - \(y\) terminally spans to \(z\) and \(y'\) initially spans to \(z\) \(\cup\)
  - \(z=y \cup z=y'\)
- Conspiracy graph: if \(\delta(y, y')\) not empty, edge from \(y\) to \(y'\)
Conspiracy theorems:

• Can_share(α,x,y,G₀) iff conspiracy path from an item in an island containing x to an item that can steal from y
• Conspirators required is shortest above path in conspiracy graph

Protection Models:
Do we have a contradiction?

• Harrison-Ruzzo-Ullman model (commands to change access control matrix
  – Safety undecidable
• Take-Grant Protection Model
  – Decidable in linear time
• What is the difference?
  – Restrictions on allowable operations
• What might we get with other sets of restrictions?
Schematic Protection Model

- Key idea: Protection Type $\tau$
  - Label that determines how control rights affect an entity
  - Take-Grant: subject and object are different protection types
  - Unix file system: File, Directory, ???

- Ticket: Describes a set of rights
  - Entity has set $\text{dom}(X)$ of tickets $Y/z$ describing $X$'s rights $z$ over entities $Y$

- Inert right vs. Control right
  - Inert right doesn’t affect protection state
Transferring Rights

- Link predicate: \( \text{link}_i(X, Y) \)
  - conjunction or disjunction of
    - \( X/z \in \text{dom}(X), X/z \in \text{dom}(Y) \)
    - \( Y/z \in \text{dom}(X), Y/z \in \text{dom}(Y) \)
    - \text{true}
  - Determines if \( X \) and \( Y \) “connected” to transfer right
  - Example: \( \text{link}(X, Y) = Y/g \in \text{dom}(X) \lor X/t \in \text{dom}(Y) \)
- Filter function: conditions on transfer
- Copy \( X/r:c \) from \( Y \) to \( Z \) allowed iff \( \exists i \) such that:
  - \( X/rc \in \text{dom}(Y) \)
  - \( \text{link}_i(Y, Z) \)
  - \( \tau(X)/r:c \in \text{filter}_i(\tau(Y), \tau(Z)) \)

Link Predicate

- Idea: \( \text{link}_i(X, Y) \) if \( X \) can assert some control right over \( Y \)
- Conjunction or disjunction of:
  - \( X/z \in \text{dom}(X) \)
  - \( X/z \in \text{dom}(Y) \)
  - \( Y/z \in \text{dom}(X) \)
  - \( Y/z \in \text{dom}(Y) \)
  - \text{true}
Examples

- Take-Grant:
  \[ \text{link}(X, Y) = Y/g \in \text{dom}(X) \lor X/t \in \text{dom}(Y) \]
- Broadcast:
  \[ \text{link}(X, Y) = X/b \in \text{dom}(X) \]
- Pull:
  \[ \text{link}(X, Y) = Y/p \in \text{dom}(Y) \]

Filter Function

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

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

Example

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

- Must handle type, tickets of new entity
- Relation can\textbullet\text{create}(a, b)
  - Subject of type $a$ can create entity of type $b$
- Rule of acyclic creates:

\[
\begin{array}{c}
  a \\ \downarrow \\
  c \\
\end{array} \rightarrow \begin{array}{c}
  b \\ \downarrow \\
  d \\
\end{array}
\]

\[
\begin{array}{c}
  a \\ \downarrow \\
  c \\
\end{array} \rightarrow \begin{array}{c}
  b \\ \downarrow \\
  d \\
\end{array}
\]

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. $a/r.c \in cr_P(a, b) \Rightarrow self/r.c \in cr_P(a, b)$
Example: File Permissions

- Types: *users*, *files*
- *(Inert) Rights: \{ r:c, w:c, x:c \}*
  - read, write, execute; copy on each
- \( \forall U, V \in \text{users}, link(U, V) = \text{true} \)
  - Anyone can grant a right to anyone if they posses the right to do so (copy)
- \( f(\text{user}, \text{user}) = \{ \text{file/r, file/w, file/x} \} \)
  - Can copy read, write, execute
  - *But not copy right*

Safety Analysis in SPM

- Idea: derive *maximal state* where changes don’t affect analysis
  - Similar to determining max flow
- Theorems:
  - A maximal state exists for every system
  - If parent gives child only rights parent has (conditions somewhat more complex), can easily derive maximal state
Typed Access Matrix Model

- Finite set $T$ of types ($TS \subseteq T$ for subjects)
- Protection State: $(S, O, \tau, A)$
  - $\tau: O \rightarrow T$ is a type function
  - Operations same as Harrison-Ruzzo-Ullman except create adds type
- $\tau$ is child type iff command creates create subject/object of type $\tau$ (otherwise parent)
- If parent/child graph from all commands acyclic, then:
  - Safety is decidable
  - Safety is NP-Hard
  - Safety is polynomial if all commands limited to three parameters

Comparing Models

- Expressive Power
  - HRU/Access Control Matrix subsumes Take-Grant
  - HRU subsumes Typed Access Control Matrix
  - SPM subsumes Take-Grant
    - Subject/Object protection types
    - ticket is label on an edge
    - take/grant are control rights
- What about SPM and HRU?
  - SPM has no revocation (delete/destroy)
- HRU without delete/destroy (monotonic HRU)?
  - MTAM subsumes monotonic mono-operational HRU
  - HRU can have create requiring multiple “parents”
Extended Schematic Protection Model

- Adds “joint create”: new node has multiple parents
  - Allows more natural representation of sharing between mutually suspicious parties
    - Create joint node for sharing
  - In Take-Grant, SPM, must create two nodes, they interact to share (equivalent power)
- Monotonic ESPM and Monotonic HRU equivalent

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
  - $X_1, \ldots, X_n$ parents, $Y$ created
  - $R_{1,i}, R_{2,i}, R_3, R_{4,i} \subseteq R$
• Rules
  - $cr_{P,i}(\tau(X_1), \ldots, \tau(X_n)) = Y / R_{1,1} \cup X_i / R_{2,i}$
  - $cr_{C}(\tau(X_1), \ldots, \tau(X_n)) = Y / R_3 \cup X_1 / R_{4,1} \cup \ldots \cup X_n / R_{4,n}$

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 $x \in R$
  - $cc(a, a) = p$
  - $cr_{Anna}(a, a, p) = cr_{Bill}(a, a, p) = \emptyset$
  - $cr_{proxy}(a, a, p) = \{ Anna/x, Bill/x \}$
2-Parent Joint Create Suffices

• Goal: emulate 3-parent joint create with 2-parent joint create

• Definition of 3-parent joint create (subjects $P_1$, $P_2$, $P_3$; child $C$):
  - $cc(\tau(P_1), \tau(P_2), \tau(P_3)) = Z \subseteq T$
  - $cr_{P_1}(\tau(P_1), \tau(P_2), \tau(P_3)) = C/R_{1,1} \cup P_1/R_{2,1}$
  - $cr_{P_2}(\tau(P_1), \tau(P_2), \tau(P_3)) = C/R_{2,1} \cup P_2/R_{2,2}$
  - $cr_{P_3}(\tau(P_1), \tau(P_2), \tau(P_3)) = C/R_{3,1} \cup P_3/R_{2,3}$

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 $P_1, P_2, P_3$ have types $p_1, p_2, p_3$
- Child $C$ of type $c$
- Parent agents $A_1, A_2, A_3$ of types $a_1, a_2, a_3$
- Child agent $S$ of type $s$
- Type $t$ is parentage
  - if $X/t \in \text{dom}(Y)$, $X$ is $Y$'s parent
- Types $t, a_1, a_2, a_3, s$ are new types

Can•Create

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

- Following added to create rule:
  - \( cr_P(p_1, a_1) = \emptyset \)
  - \( cr_C(p_1, a_1) = p_1/Rtc \)
    - Agent’s parent set to creating parent; agent has all rights over parent
  - \( cr_{P_{\text{first}}}(p_2, a_1, a_2) = \emptyset \)
  - \( cr_{P_{\text{second}}}(p_2, a_1, a_2) = \emptyset \)
  - \( cr_C(p_2, a_1, a_2) = p_2/Rtc \cup a_1/tc \)
    - Agent’s parent set to creating parent and agent; agent has all rights over parent (but not over agent)

- \( cr_{P_{\text{first}}}(p_3, a_2, a_3) = \emptyset \)
- \( cr_{P_{\text{second}}}(p_3, a_2, a_3) = \emptyset \)
- \( cr_C(p_3, a_2, a_3) = p_3/Rtc \cup a_2/tc \)
  - Agent’s parent set to creating parent and agent; agent has all rights over parent (but not over agent)
- \( cr_P(a_3, s) = \emptyset \)
- \( cr_C(a_3, s) = a_3/tc \)
  - Child’s agent has third agent as parent \( cr_P(a_3, s) = \emptyset \)
- \( cr_P(s, c) = C/Rtc \)
- \( cr_C(s, c) = c/R_3 t \)
  - 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
    - \( link_1(A_1, A_2) = A_1/t \in \text{dom}(A_2) \odot A_2/t \in \text{dom}(A_2) \)
    - \( link_1(A_2, A_3) = A_2/t \in \text{dom}(A_3) \odot A_3/t \in \text{dom}(A_3) \)
    - \( link_2(S, A_3) = A_3/t \in \text{dom}(S) \odot C/t \in \text{dom}(C) \)
    - \( link_3(A_1, C) = C/t \in \text{dom}(A_1) \)
    - \( link_3(A_2, C) = C/t \in \text{dom}(A_2) \)
    - \( link_3(A_3, C) = C/t \in \text{dom}(A_3) \)
    - \( link_4(A_1, P_1) = P_1/t \in \text{dom}(A_1) \odot A_1/t \in \text{dom}(A_1) \)
    - \( link_4(A_2, P_2) = P_2/t \in \text{dom}(A_2) \odot A_2/t \in \text{dom}(A_2) \)
    - \( link_4(A_3, P_3) = P_3/t \in \text{dom}(A_3) \odot A_3/t \in \text{dom}(A_3) \)

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 $A_1$, $A_2$, $A_3$, $S$, $C$; then

- $P_1$ has no relevant tickets
- $P_2$ has no relevant tickets
- $P_3$ has no relevant tickets
- $A_1$ has $P_1/Rtc$
- $A_2$ has $P_2/Rtc \cup A_1/tc$
- $A_3$ has $P_3/Rtc \cup A_2/tc$
- $S$ has $A_3/tc \cup C/Rtc$
- $C$ has $C/R_3$

---

**Construction**

- Only $link_2(S, A_3)$ true $\Rightarrow$ apply $f_2$
  - $A_3$ has $P_3/Rtc \cup A_2/tc \cup A_3/tc \cup C/Rtc$
- Now $link_1(A_3, A_2)$ true $\Rightarrow$ apply $f_1$
  - $A_2$ has $P_2/Rtc \cup A_2/tc \cup A_2/tc \cup C/Rtc$
- Now $link_1(A_2, A_1)$ true $\Rightarrow$ apply $f_1$
  - $A_1$ has $P_2/Rtc \cup A_1/tc \cup A_1/tc \cup C/Rtc$
- Now all $link_3$s true $\Rightarrow$ apply $f_3$
  - $C$ has $C/R_3 \cup P_1/R_{4,1} \cup P_2/R_{4,2} \cup P_3/R_{4,3}$
Finish Construction

• Now \( \text{link}_4 \) is true \( \Rightarrow \) apply \( f_4 \)
  
  – \( P_1 \) has \( C/R_{1,1} \cup P_1/R_{2,1} \)
  
  – \( P_2 \) has \( C/R_{1,2} \cup P_2/R_{2,2} \)
  
  – \( P_3 \) has \( C/R_{1,3} \cup P_3/R_{2,3} \)

• 3-parent joint create gives same rights to \( P_1, P_2, P_3, C \)

• If create of \( C \) fails, \( \text{link}_2 \) 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
Example: 3-Parent Joint Creation

- Simulate with 2-parent
  - Nodes $P_1$, $P_2$, $P_3$ parents
  - Create node $C$ with type $c$ with edges of type $e$
  - Add node $A_1$ of type $a$ and edge from $P_1$ to $A_1$ 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:
  - $P_1 \rightarrow A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow S \rightarrow C$: $P_1$ to $C$ edge type $e$
  - $P_2 \rightarrow A_2 \rightarrow A_3 \rightarrow S \rightarrow C$: $P_2$ to $C$ edge type $e$
  - $P_3 \rightarrow A_3 \rightarrow S \rightarrow C$: $P_3$ 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 $P_1$, $P_2$, $P_3$, $C$ are of same type as in $TWO$, and edges from $P_1$, $P_2$, $P_3$ to $C$ are of type $e$, and no types $a$ and $e'$ exist in $TWO$
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
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 $X_1$, $X_2$, $X_3$
  – 2-parent joint create
  – 1 node type, 1 edge type
  – No edge adding operations
  – Initial state: $X_1$, $X_2$, $X_3$, 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 $X_1, X_2$ jointly create $Y$ in $A$
  - Edges from $X_1, X_2$ to $Y$, no edge from $X_3$ to $Y$
- Can $B$ simulate this?
  - Without loss of generality, $X_1$ creates $Y$
  - Must have edge adding operation to add edge from $X_2$ 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 $X_2$ to $C$ can add one from $X_3$ 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 $(S, O, \tau, A)$, where $\tau: O \rightarrow T$ specifies type of each object
  - If $X$ subject, $\tau(X)$ in $TS$
  - If $X$ object, $\tau(X)$ 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 \not\in 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] = \emptyset], (\forall x \in S')[a'[x, s] = \emptyset]
  – (\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]
Create Object

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

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 \( \alpha \) if any of create subject \( x_i \) of type \( t_i \) or create object \( x_i \) of type \( t_i \) occur in \( \alpha \)
  - \( t_i \) parent type otherwise
Cyclic Creates

**command** `havoc(s_1 : u, s_2 : u, o_1 : v, o_2 : v, o_3 : w, o_4 : 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