Theorem: Can_share(α,x,y,G₀) (for subjects)

- Subject_can_share(α, x, y, G₀) is true iff x and y are subjects and
  - there is an α edge from x to y in G₀
  OR if:
    - ∃ a subject s ∈ G₀ with an s-to-y α edge, and
    - ∃ islands I₁, …, Iₙ such that x ∈ I₁, s ∈ Iₙ, and there is a bridge from I_j to I_j₊₁

Diagram showing the relationships with islands and bridges.
What about objects?
Initial, terminal spans

- **x initially spans** to y if x is a subject and there is a tg-path associated with word \{t \rightarrow *g \rightarrow \} between them
  - x can grant a right to y
- **x terminally spans** to y if x is a subject and there is a tg-path associated with word \{t \rightarrow *\} between them
  - 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 I₁ to Iₙ⁺¹

\[ x' \text{ can grant a right to } x \quad s' \text{ can take a right from } s \]
**Theorem:** \text{Can\_share}(\alpha, x, y, G_0)

- **Corollary:** There is an $O(|V|+|E|)$ algorithm to test can\_share: Decidable in linear time!!
- **Theorem**
  - Let $G_0$ contain exactly one vertex and no edges,
  - $R$ a set of rights.
  - $G_0 \vdash^* G$ iff $G$ is a finite directed acyclic graph, with edges labeled from $R$, and at least one subject with no incoming edge.
  - **Only if** part: $v$ is initial subject and $G_0 \vdash^* G$;
    - No rule allows the deletion of a vertex
    - No rule allows an incoming edge to be added to a vertex without any incoming edges. Hence, as $v$ has no incoming edges, it cannot be assigned any

**Theorem:** \text{Can\_share}(\alpha, x, y, G_0)

- **If** part: $G$ meets the requirement
  - Assume $v$ is the vertex with no incoming edge and apply rules
  1. Perform “$v$ creates ($\alpha \cup \{g\}$ to) new $x_i$” for all $2 \leq i \leq n$, and $\alpha$ is union of all labels on the incoming edges going into $x_i$ in $G$
  2. For all pairs $x$, $y$ with $x \alpha$ over $y$ in $G$, perform “$v$ grants ($\alpha$ to $y$) to $x$”
  3. If $\beta$ is the set of rights $x$ has over $y$ in $G$, perform “$v$ removes ($\alpha \cup \{g\} - \beta$) to $y$”
Example

Take-Grant Model: Sharing through a Trusted Entity

- Let $p$ and $q$ be two processes
- Let $b$ be a buffer that they share to communicate
- Let $s$ be a third party (e.g., operating system) that controls $b$

Witness

- $S$ creates ($r$, $w$, to new object) $b$
- $S$ grants ($r$, $w$, $b$) to $p$
- $S$ grants ($r$, $w$, $b$) to $q$
Theft in Take-Grant Model

- Can_steal(α,x,y,G₀) is true if there is no α edge from x to y in G₀ and ∃ sequence G₁, ..., Gₙ s. t.:
  - ∃ α edge from x to y in Gₙ,
  - ∃ rules ρ₁,..., ρₙ that take Gᵢ⁻¹ ⊢ ρᵢ Gᵢ, and
  - ∀ v,w ∈ Gᵢ, 1 ≤ i < n, if ∃ α edge from v to y in G₀ then ρᵢ is not “v grants (α to y) to w”

- Disallows owners of α rights to y from transferring those rights
- Does not disallow them to transfer other rights
- This models a Trojan horse

A witness to theft

- u grants (t to v) to s
- s takes (t to u) from v
- s takes (α to w) from u
Theorem: When Theft Possible

\[ \text{Can\_steal}(\alpha, x, y, G_0) \text{ iff there is no } \alpha \text{ edge from } x \text{ to } y \text{ in } G_0 \text{ and } \exists G_1, \ldots, G_n \text{ s.t.:} \]

- There is no \( \alpha \) edge from \( x \) to \( y \) in \( G_0 \).
- \( \exists \) subject \( x' \) such that \( x' = x \) or \( x' \) initially spans to \( x \), and
- \( \exists s \) with \( \alpha \) edge to \( y \) in \( G_0 \) and \( \text{can\_share}(t, x, s, G_0) \)

Proof:

- \( \Rightarrow \): Assume the three conditions hold
  - \( x \) can get \( \alpha \) right over \( s \) (\( x \) is a subject) and then take \( \alpha \) right over \( y \) from \( s \)
  - \( x' \) creates a surrogate to pass \( \alpha \) to \( x \) (\( x \) is an object)
    - \( x' \) initially spans to \( x \) (Theorem 3.10 - \( \text{can\_share}(t, x', s, G_0) \))

- \( \Leftarrow \): Assume \( \text{can\_steal} \) is true:
  - No \( \alpha \) edge from definition 3.10 in \( G_0 \).
  - \( \text{Can\_share}(\alpha, x, y, G_0) \) from definition 3.10 condition (a): \( \alpha \) from \( x \) to \( y \) in \( G_n \)
  - \( s \) exists from \( \text{can\_share} \) and earlier theorem
  - Show \( \text{Can\_share}(t, x, s, G_0) \) holds: \( s \) can’t grant \( \alpha \) (definition), someone else must get \( \alpha \) from \( s \), show that this can only be accomplished with take rule
Theft indicates cooperation: which subjects are actors in a transfer of rights, and which are not?

Next question is

How many subjects are needed to enable $\text{Can\_share}(\alpha, x, y, G_0)$?

Note that a vertex $y$

- Can take rights from any vertex to which it terminally spans
- Can pass rights to any vertex to which it initially spans

Access set $A(y)$ with focus $y$ (y is subject) is union of

- set of vertices $y$,
- vertices to which $y$ initially spans, and
- 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 $z$
- $y$ terminally spans to $z$ and $y'$ initially spans to $z$
- $z = y$ & $z = y'$

Conspiracy graph $H$ of $G_0$:

- Represents the paths along which subjects can transfer rights
- For each subject in $G_0$, there is a corresponding vertex $h(x)$ in $H$
- if $\delta(y, y')$ not empty, edge from $h(y)$ to $h(y')$
Example

Theorems

- \( I(p) = \) contains the vertex \( h(p) \) and the set of all vertices \( h(p') \) such that \( p' \) initially spans to \( p \)
- \( T(q) = \) contains the vertex \( h(q) \) and the set of all vertices \( h(q') \) such that \( q' \) terminally spans to \( q \)
- **Theorem 3-13:**
  - \( \text{Can}_\text{share}(\alpha, x, y, G_0) \) iff there is a path from some \( h(p) \) in \( I(x) \) to some \( h(q) \) in \( T(y) \)
- **Theorem 3-14:**
  - Let \( L \) be the number of vertices on a shortest path between \( h(p) \) and \( h(q) \) (as in theorem 3-13), then \( L \) conspirators are necessary and sufficient to produce a witness to \( \text{Can}_\text{share}(\alpha, x, y, G_0) \)
Back to HRU:
Fundamental questions

- How can we determine that a system is secure?
  - Need to define what we mean by a system being “secure”
- Is there a generic algorithm that allows us to determine whether a computer system is secure?

Turing Machine & halting problem

- The halting problem:
  - Given a description of an algorithm and a description of its initial arguments, determine whether the algorithm, when executed with these arguments, ever halts (the alternative is that it runs forever without halting).
- Reduce TM to Safety problem
  - If Safety problem is decidable then it implies that TM halts (for all inputs) – showing that the halting problem is decidable (contradiction)
Turing Machine

- TM is an abstract model of computer
  - Alan Turing in 1936
- TM consists of
  - A tape divided into cells; infinite in one direction
  - A set of tape symbols $M$
    - $M$ contains a special blank symbol $b$
  - A set of states $K$
  - A head that can read and write symbols
  - An action table that tells the machine
    - What symbol to write
    - How to move the head ('L' for left and 'R' for right)
    - What is the next state

The action table describes the transition function
- Transition function $\delta(k, m) = (k', m', L)$:
  - in state $k$, symbol $m$ on tape location is replaced by symbol $m'$,
  - head moves to left one square, and TM enters state $k'$
- Halting state is $q_f$
  - TM halts when it enters this state
Turing Machine

Let $\delta(k, C) = (k_1, X, R)$ where $k_1$ is the next state

Current state is $k$
Current symbol is $C$

General Safety Problem

- Theorem: It is undecidable if a given state of a given protection system is safe for a given generic right
- Proof: Reduce TM to safety problem
  - Symbols, States $\Rightarrow$ rights
  - Tape cell $\Rightarrow$ subject
  - Cell $s_i$ has $A$ $\Rightarrow$ $s_i$ has $A$ rights on itself
  - Cell $s_k$ $\Rightarrow$ $s_k$ has end rights on itself
  - State $p$, head at $s_i$ $\Rightarrow$ $s_i$ has $p$ rights on itself
  - Distinguished Right own:
    - $s_i$ owns $s_{i+1}$ for $1 \leq i < k$
Current state is $k$
Current symbol is $C$

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
A & B & C & D \\
\end{array}
\]

\[
\begin{array}{cccc}
& s_1 & s_2 & s_3 & s_4 \\
\hline
s_1 & A & \text{own} & & \\
\hline
s_2 & B & \text{own} & & \\
\hline
s_3 & C & k & \text{own} & \\
\hline
s_4 & & D & \text{end} & \\
\end{array}
\]

**Command Mapping (Left move)**

$\delta(k, C) = (k_1, X, L)$

**command** $c_{k,C}(s_i, s_{i-1})$

if own in $a[s_{i-1}, s_i]$ and $k$ in $a[s_j, s_l]$ and $C$ in $a[s_r, s_t]$
then
- delete $k$ from $A[s_r, s_t]$;
- delete $C$ from $A[s_r, s_t]$;
- enter $X$ into $A[s_r, s_t]$;
- enter $k_1$ into $A[s_{i-1}, s_{i-1}]$;
end
Mapping (Left Move)

After $\delta(k, C) = (k_1, X, L)$ where $k$ is the current state and $k_1$ the next state.

Mapping (Initial)

Current state is $k$
Current symbol is $C$
Command Mapping
(Right move)

\[ \delta(k, C) = (k_1, X, R) \]

**command** \(c_{k,C}(s_i, s_{i+1})\)

*if own in \(a[s_i, s_{i+1}]\) and \(k\) in \(a[s_i, s_j]\) and \(C\) in \(a[s_i, s_j]\)*

*then*

- delete \(k\) from \(A[s_i, s_j]\);
- delete \(C\) from \(A[s_i, s_j]\);
- enter \(X\) into \(A[s_i, s_j]\);
- enter \(k_1\) into \(A[s_{i+1}, s_{i+1}]\);
*end*

---

Mapping

After \(\delta(k, C) = (k_1, X, R)\)

where \(k\) is the current state and \(k_1\) the next state

<table>
<thead>
<tr>
<th>(A)</th>
<th>(B)</th>
<th>(X)</th>
<th>(D)</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_1)</td>
<td>(s_2)</td>
<td>(s_3)</td>
<td>(s_4)</td>
<td></td>
</tr>
<tr>
<td>(s_1)</td>
<td>(A)</td>
<td>(own)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s_2)</td>
<td>(B)</td>
<td>(own)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s_3)</td>
<td>(X)</td>
<td>(own)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s_4)</td>
<td>(D)</td>
<td>(k_1)</td>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>
\[ \delta(k_1, D) = (k_2, Y, R) \text{ at end becomes} \]

**Command Mapping (Rightmost move)**

\[ \text{command } \text{crightmost}_{C(s_i,s_{i+1})} \]

\[ \text{if } \text{end in } a[s_i,s_{i}] \text{ and } k_1 \text{ in } a[s_i,s_{i}] \text{ and } D \text{ in } a[s_i,s_{i}] \]

\[ \text{then} \]

\[ \begin{align*}
\text{delete end from } & a[s_i,s_{i}] ; \\
\text{create subject } & s_{i+1} ; \\
\text{enter } & \text{own into } a[s_{i+1},s_{i+1}] ; \\
\text{enter end into } & a[s_{i+1},s_{i+1}] ; \\
\text{delete } & k_1 \text{ from } a[s_{i+1},s_{i+1}] ; \\
\text{delete } & D \text{ from } a[s_{i+1},s_{i+1}] ; \\
\text{enter } & Y \text{ into } a[s_{i+1},s_{i+1}] ; \\
\text{enter } & k_2 \text{ into } A[s_{i+1},s_{i+1}] ; \\
\text{end} 
\end{align*} \]

After \( \delta(k_1, D) = (k_2, Y, R) \) where \( k_1 \) is the current state and \( k_2 \) the next state

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>X</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>A</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>B</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>X</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s5</td>
<td></td>
<td></td>
<td>b k_2</td>
<td>end</td>
</tr>
</tbody>
</table>
```
Rest of Proof

- Protection system exactly simulates a TM
  - Exactly 1 end right in ACM
  - 1 right corresponds to a state
  - Thus, at most 1 applicable command in each configuration of the TM
- If TM enters state $q_{r}$, then right has leaked
- If safety question decidable, then represent TM as above and determine if $q_{r}$ leaks
  - Leaks halting state $\Rightarrow$ halting state in the matrix $\Rightarrow$ Halting state reached
- Conclusion: safety question undecidable

Other theorems

- Set of unsafe systems is recursively enumerable
  - Recursively enumerable?
- For protection system without the create primitives, (i.e., delete create primitive); the safety question is complete in P-SPACE
- It is undecidable whether a given configuration of a given monotonic protection system is safe for a given generic right
  - Delete destroy, delete primitives;
  - The system becomes monotonic as they only increase in size and complexity
Other theorems

- The safety question for biconditional monotonic protection systems is undecidable
- The safety question for monoconditional, monotonic protection systems is decidable
- The safety question for monoconditional protection systems with create, enter, delete (and no destroy) is decidable.

Observations
- Safety is undecidable for the generic case
- Safety becomes decidable when restrictions are applied

Schematic Protection Model

- Key idea is to use the notion of a protection type
  - Label that determines how control rights affect an entity
  - Take-Grant:
    - subject and object are different protection types
  - TS and TO represent subject type set and object set
  - \( \tau(\text{X}) \) is the type of entity \( \text{X} \)
- A ticket describes a right
  - Consists of an entity name and a right symbol; \( \text{X}/z \)
    - Possessor of the ticket \( \text{X}/z \) has right \( r \) over entity \( \text{X} \)
    - \( Y \) has tickets \( \text{X}/r, \text{X}/w \rightarrow Y \) has tickets \( \text{X}/nw \)
  - Each entity \( \text{X} \) has a set \( \text{dom}(\text{X}) \) of tickets \( \text{Y}/z \)
  - \( \tau(\text{X}/rc) = \tau(\text{X})/rc \) is the type of a ticket
Schematic Protection Model

- Inert right vs. Control right
  - Inert right doesn’t affect protection state, e.g. read right
  - take right in Take-Grant model is a control right
- Copy flag c
  - Every right r has an associated copyable right rc
  - r:c means r or rc
- Manipulation of rights
  - A link predicate
    - Determines if a source and target of a transfer are “connected”
  - A filter function
    - Determines if a transfer is authorized

Transferring Rights

- \( \text{dom}(X) \) : set of tickets that X has
- Link predicate: \( \text{link}(X, Y) \)
  - conjunction or disjunction of the following terms
    - \( X/z \in \text{dom}(X) \); \( Y/z \in \text{dom}(Y) \);
    - \( Y/z \in \text{dom}(X) \); \( Y/z \in \text{dom}(Y) \)
    - \text{true}
  - Determines if X and Y ”connected” to transfer right
  - Examples:
    - Take-Grant: \( \text{link}(X, Y) = Y/g \in \text{dom}(X) v X/t \in \text{dom}(Y) \)
    - Broadcast: \( \text{link}(X, Y) = X/b \in \text{dom}(X) \)
    - Pull: \( \text{link}(X, Y) = Y/p \in \text{dom}(Y) \)
    - Universal: \( \text{link}(X, Y) = \text{true} \)
- Scheme: a finite set of link predicates is called a scheme
Filter Function

- Filter function:
  - Imposes conditions on when tickets can be transferred
  - $f: TS \times TS \rightarrow 2^{TV}$ (range is copyable rights)
- $X/r:c$ can be copied from $dom(Y)$ to $dom(Z)$ if $\exists i$ s. t. the following are true:
  - $X/r:c \in dom(Y)$
  - $link(Y, Z)$
  - $\tau(X)/r:c \in f(\tau(Y), \tau(Z))$
- Examples:
  - If $f(\tau(Y), \tau(Z)) = T \times R$ then any rights are transferable
  - If $f(\tau(Y), \tau(Z)) = T \times RI$ then only inert rights are transferable
  - If $f(\tau(Y), \tau(Z)) = \emptyset$ then no tickets are transferable
- One filter function is defined for each link predicate

SCM Example 1

- Owner-based policy
  - Subject $U$ can authorize subject $V$ to access an object $F$ if $U$ owns $F$
  - Types: $TS=\{user\}$, $TO=\{file\}$
  - Ownership is viewed as copy attributes
  - If $U$ owns $F$, all its tickets for $F$ are copyable
  - $RI: \{r:c, w:c, a:c, x:c\}$; $RC$ is empty
    - read, write, append, execute; copy on each
  - $\forall U, V \in user$, $link(U, V) = true$
    - Anyone can grant a right to anyone else if they posses the right to do so (copy)
  - $f(user, user) = \{filer, filew, filea, filex\}$
    - Can copy read, write, append, execute
SPM Example 1

- Peter owns file Doom; can he give Paul execute permission over Doom?
  1. \( \tau(\text{Peter}) \) is user and \( \tau(\text{Paul}) \) is user
  2. \( \tau(\text{Doom}) \) is file
  3. \( \text{Doom}/x_c \in \text{dom}(\text{Peter}) \)
  4. \( \text{Link} (\text{Peter}, \text{Paul}) = \text{TRUE} \)
  5. \( \tau(\text{Doom})/x \in f(\tau(\text{Peter}), \tau(\text{Paul})) \) - because of 1 and 2

Therefore, Peter can give ticket \( \text{Doom}/x_c \) to Paul

SPM Example 2

- Take-Grant Protection Model
  - \( TS = \{ \text{subjects} \}, TO = \{ \text{objects} \} \)
  - \( RC = \{ tc, gc \}, RI = \{ rc, wc \} \)
  - Note that all rights can be copied in T-G model
  - \( \text{link}(p, q) = p/t \in \text{dom}(q) \lor q/t \in \text{dom}(p) \)
  - \( f(\text{subject, subject}) = \{ \text{subject, object} \} \times \{ tc, gc, rc, wc \} \)
  - Note that any rights can be transferred in T-G model
Demand

- A subject can demand a right from another entity
  - Demand function $d: TS \rightarrow 2^{TxR}$
  - Let $a$ and $b$ be types
    - $a/r: c \in d(b)$: every subject of type $b$ can demand a ticket $X/r: c$ for all $X$ such that $\tau(X) = a$
  - A sophisticated construction eliminates the need for the demand operation – hence omitted

Create Operation

- Need to handle
  - type of the created entity, &
  - tickets added by the creation
- Relation $can\cdot create(a, b) \subseteq TS \times T$
  - A subject of type $a$ can create an entity of type $b$
- Rule of acyclic creates
  - Limits the membership in $can\cdot create(a, b)$
  - If a subject of type $a$ can create a subject of type $b$, then none of the descendants can create a subject of type $a$
Create operation
Distinct Types

- create rule \( cr(a, b) \) specifies the
  - tickets introduced when a subject of type \( a \) creates an
    entity of type \( b \)
- \( B \) object: \( cr(a, b) \subseteq \{ b/r:c \in RI \} \)
  - Only inert rights can be created
  - \( A \) gets \( B/r:c \) iff \( b/r:c \in cr(a, b) \)
- \( B \) 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 the created
  - \( cr(a, a) = \{ a/r:c, self/r:c | r.c \in R \} \)
  - \( cr(a, a) = cr_c(a, b) | cr_p(a, b) \) is attenuating if:
    1. \( cr_c(a, b) \subseteq cr_p(a, b) \) and
    2. \( a/r:c \in cr_p(a, b) \Rightarrow self/r:c \in cr_p(a, b) \)
- A scheme is attenuating if,
  - For all types \( a, cc(a, a) \rightarrow cr(a, a) \) is attenuating
Examples

- Owner-based policy
  - Users can create files: $cc(\text{user}, \text{file})$ holds
  - Creator can give itself any inert rights: $cr(\text{user}, \text{file}) = \{\text{file}/r.c| r \in R\}$

- Take-Grant model
  - A subject can create a subject or an object
    - $cc(\text{subject}, \text{subject})$ and $cc(\text{subject}, \text{object})$ hold
  - Subject can give itself any rights over the vertices it creates but the subject does not give the created subject any rights (although grant can be used later)
    - $cr(\text{a}, \text{b}) = \emptyset$; $cr(\text{a}, \text{b}) = \{\text{sub/tc, sub/gc, sub/rc, sub/wc}\}$
    - Hence,
      - $cr(\text{sub}, \text{sub}) = \{\text{sub/tc, sub/gc, sub/rc, sub/wc}\} | \emptyset$
      - $cr(\text{sub}, \text{obj}) = \{\text{obj/tc, obj/gc, obj/rc, obj/wc}\} | \emptyset$

Safety Analysis in SPM

- Idea: derive maximal state where changes don’t affect analysis
  - Indicates all the tickets that can be transferred from one subject to another
  - Indicates what the maximum rights of a subject is in a system

- 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
  - Safety: If the scheme is acyclic and attenuating, the safety question is decidable
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 in HRU model except create adds type
- $\tau$ is child type iff command create creates subject/object of type $\tau$
- 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

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 does not
  - SPM cannot express multiparent creates easily, and not at all if the parents are of different types because $\text{can}\cdot\text{create}$ allows for only one type of creator
  - Suggests SPM is less expressive than HRU
Comparing Models

- Expressive Power
  - HRU/Access Control Matrix subsumes Take-Grant
  - HRU subsumes Typed Access Control Matrix
  - SPM subsumes
    - Take-Grant
    - Multilevel security
    - Integrity models
- What about SPM and HRU?
  - SPM has no revocation (delete/destroy)
- HRU without delete/destroy (monotonic HRU)
  - MTAM subsumes monotonic mono-operational HRU

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
- Monotonic ESPM and Monotonic HRU are equivalent