Safety Question

• In this model:
  Is it possible to have a derivable state with $X/r:c$ in $\text{dom}(A)$, or does there exist a subject $B$ with ticket $X/rc$ in the initial state or which can demand $X/rc$ and $\tau(X)/r:c$ in $\text{flow}^*(B,A)$?

• 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 $\sigma: \text{ENT}^h \rightarrow \text{ENT}^h$ by:
  • if $X$ in $\text{ENT}^0$, then $\sigma(X) = X$
  • if $Y$ creates $X$ and $\tau(Y) = \tau(X)$, then $\sigma(X) = \sigma(Y)$
  • if $Y$ creates $X$ and $\tau(Y) \neq \tau(X)$, then $\sigma(X) = \tau(Y)$-surrogate of $\sigma(Y)$
Implications

• $\tau(\sigma(X)) = \tau(X)$
• If $\tau(X) = \tau(Y)$, then $\sigma(X) = \sigma(Y)$
• If $\tau(X) \neq \tau(Y)$, then
  • $\sigma(X)$ creates $\sigma(Y)$ in the construction of $u$
  • $\sigma(X)$ creates entities $X'$ of type $\tau(X') = \tau(\sigma(X))$
• From these, for a system with an acyclic attenuating scheme, if $X$ creates $Y$, then tickets that would be introduced by pretending that $\sigma(X)$ creates $\sigma(Y)$ are in $dom^u(\sigma(X))$ and $dom^u(\sigma(Y))$
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 “$\sigma(X)$ demands $\sigma(Y)/r:c$”
  • “$Y$ copies $X/r:c$ from $Y$” becomes “$\sigma(Y)$ copies $\sigma(X)/r:c$ from $\sigma(Y)$”
Tickets in Parallel

• Lemma
  • All transitions in $G$ legal; if $X/r:c \in \text{dom}^h(Y)$, then $\sigma(X)/r:c \in \text{dom}^h(\sigma(Y))$

• 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 \( \mathbf{X}/r:c \) to be in \( \text{dom}^h(\mathbf{Y}) \):
  • \( \mathbf{X}/r:c \in \text{dom}^0(\mathbf{Y}) \) means \( \mathbf{X}, \mathbf{Y} \in \text{ENT}^0 \), so trivially \( \sigma(\mathbf{X})/r:c \in \text{dom}^g(\sigma(\mathbf{Y})) \) holds
  • A create added \( \mathbf{X}/r:c \in \text{dom}^h(\mathbf{Y}) \): previous lemma says \( \sigma(\mathbf{X})/r:c \in \text{dom}^g(\sigma(\mathbf{Y})) \) holds
  • A demand added \( \mathbf{X}/r:c \in \text{dom}^h(\mathbf{Y}) \): corresponding demand operation in \( G \) gives \( \sigma(\mathbf{X})/r:c \in \text{dom}^g(\sigma(\mathbf{Y})) \)
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 $X/r:c \in dom^h(Z)$
  • Construction of $G$ means final operation is $\sigma(X)/r:c \in dom^g(\sigma(Y))$
• Proves second part of claim
Step

• $H$’legal, so for $H$ to be legal, we have:
  1. $X/rc \in dom^h(Y)$
  2. $link_i^h(Y, Z)$
  3. $\tau(X/r:c) \in f_i(\tau(Y), \tau(Z))$

• By IH, 1, 2, as $X/r:c \in dom^h(Y)$,
  $\sigma(X)/r:c \in dom^g(\sigma(Y))$ and $link_i^g(\sigma(Y), \sigma(Z))$

• As $\sigma$ preserves type, IH and 3 imply
  $\tau(\sigma(X)/r:c) \in f_i(\tau((\sigma(Y)), \tau(\sigma(Z)))$

• IH says $G$’legal, so $G$ is legal
Corollary

• If $\text{link}_i^h(X, Y)$, then $\text{link}_i^g(\sigma(X), \sigma(Y))$
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

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

• 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 $path^h(X,Y)$ has corresponding $path^g(\sigma(X), \sigma(Y))$ such that $cap(path^h(X,Y)) = cap(path^g(\sigma(X), \sigma(Y)))$
  • 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 $path^h(X, Y) = 1$. By definition of $path^h$, $link^h_i(X, Y)$, hence $link^g_i(\sigma(X), \sigma(Y))$. As $\sigma$ preserves type, this means

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

• Now assume this is true when $path^h(X, Y)$ has length $k$
Step

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

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

• Let maximal state corresponding to $v$ be $\#u$
  • Deriving history has no creates
  • By theorem,

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

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

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

• So $\#u$ is maximal state for system with acyclic attenuating scheme
  • $\#u$ derivable from $u$ in time polynomial to $|\text{SUB}^u|$   • Worst case computation for $\text{flow}^\#u$ 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\(\text{\textbullet\text{create}}\) 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
  • \(X_1, \ldots, X_n\) 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\textbullet create

- Following added to \textit{can\textbullet create}:
  - \textit{cc}(p_1) = a_1
  - \textit{cc}(p_2, a_1) = a_2
  - \textit{cc}(p_3, a_2) = a_3
    - Parents creating their agents; note agents have maximum of 2 parents
  - \textit{cc}(a_3) = s
    - Agent of all parents creates agent of child
  - \textit{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
  • $cr_{Pfirst}(p_2, a_1, a_2) = \emptyset$
  • $cr_{Psecond}(p_2, a_1, a_2) = \emptyset$
  • $cr_C(p_2, a_1, a_2) = p_2/Rtc \cup a_1/tc$
    • Agent’s parent set to creating parent and agent; agent has all rights over parent (but not over agent)
Creation Rules

- \( cr_{P\text{first}}(p_3, a_2, a_3) = \emptyset \)
- \( cr_{P\text{second}}(p_3, a_2, a_3) = \emptyset \)
- \( cr_C(p_3, a_2, a_3) = p_3/Rtc \cup a_2/tc \)
  - 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 \( cr_p(a_3, s) = \emptyset \)
- \( cr_p(s, c) = C/Rtc \)
- \( cr_C(s, c) = c/R_3t \)
  - 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
    • \text{link}_1(A_2, A_1) = A_1/t \in \text{dom}(A_2) \land A_2/t \in \text{dom}(A_2)
    • \text{link}_1(A_3, A_2) = A_2/t \in \text{dom}(A_3) \land A_3/t \in \text{dom}(A_3)
    • \text{link}_2(S, A_3) = A_3/t \in \text{dom}(S) \land C/t \in \text{dom}(C)
    • \text{link}_3(A_1, C) = C/t \in \text{dom}(A_1)
    • \text{link}_3(A_2, C) = C/t \in \text{dom}(A_2)
    • \text{link}_3(A_3, C) = C/t \in \text{dom}(A_3)
    • \text{link}_4(A_1, P_1) = P_1/t \in \text{dom}(A_1) \land A_1/t \in \text{dom}(A_1)
    • \text{link}_4(A_2, P_2) = P_2/t \in \text{dom}(A_2) \land A_2/t \in \text{dom}(A_2)
    • \text{link}_4(A_3, P_3) = P_3/t \in \text{dom}(A_3) \land A_3/t \in \text{dom}(A_3)
Filter Functions

\[
\begin{align*}
 f_1(a_2, a_1) &= a_1 / t \cup c / Rtc \\
 f_1(a_3, a_2) &= a_2 / t \cup c / Rtc \\
 f_2(s, a_3) &= a_3 / t \cup c / Rtc \\
 f_3(a_1, c) &= p_1 / R_{4,1} \\
 f_3(a_2, c) &= p_2 / R_{4,2} \\
 f_3(a_3, c) &= p_3 / R_{4,3} \\
 f_4(a_1, p_1) &= c / R_{1,1} \cup p_1 / R_{2,1} \\
 f_4(a_2, p_2) &= c / R_{1,2} \cup p_2 / R_{2,2} \\
 f_4(a_3, p_3) &= c / R_{1,3} \cup p_3 / R_{2,3}
\end{align*}
\]
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 $C/R_{3t}$
Construction

• Only $link_2(S, A_3)$ true $\Rightarrow$ apply $f_2$
  • $A_3$ has $P_3/Rtc \cup A_2/t \cup A_3/t \cup C/Rtc$

• Now $link_1(A_3, A_2)$ true $\Rightarrow$ apply $f_1$
  • $A_2$ has $P_2/Rtc \cup A_1/tc \cup A_2/t \cup C/Rtc$

• Now $link_1(A_2, A_1)$ true $\Rightarrow$ apply $f_1$
  • $A_1$ has $P_2/Rtc \cup A_1/tc \cup C/Rtc$

• 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 $link_4$ 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, $link_2$ 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 $P_1$, $P_2$, $P_3$ 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 \textit{TWO}

• Equivalent to 3-parent joint creation scheme \textit{THREE} in which \(P_1, P_2, P_3, C\) are of same type as in \textit{TWO}, and edges from \(P_1, P_2, P_3\) to \(C\) are of type \(e\), and no types \(a\) and \(e'\) exist in \textit{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 $X_1, X_2, X_3$
  • 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

• 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

• $X_1, X_2$ nodes in $A$
  • They create $Y_1, Y_2, Y_3$ using multiparent create rule
  • $Y_1, Y_2$ create $Z$, again using multiparent create rule
  • Note: no edge from $Y_3$ to $Z$ can be added, as $A$ has no edge-adding operation
Outline of Proof

- \( \mathbf{W}, \mathbf{X}_1, \mathbf{X}_2 \) nodes in \( B \)
  - \( \mathbf{W} \) 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 \( \mathbf{Z} \), again using single parent create rule; now must add edge from \( \mathbf{Y}_2 \) to \( \mathbf{Z} \) to simulate \( A \)
  - Use same edge adding rule to add edge from \( \mathbf{Y}_3 \) to \( \mathbf{Z} \): 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 $(S, O, \tau, A)$
  • $\tau: O \rightarrow T$ 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: \( s \notin S \)

• Primitive command: \textbf{create subject} \( s \) \textbf{of type} \( t \)

• Postconditions:
  • \( S' = S \cup \{ s \}, \ O' = O \cup \{ s \} \)
  • \((\forall y \in O)[\tau'(y) = \tau(y)], \ \tau'(s) = t\)
  • \((\forall y \in O')[\alpha'[s, y] = \emptyset], \ (\forall x \in S')[\alpha'[x, s] = \emptyset]\)
  • \((\forall x \in S)(\forall y \in O)[\alpha'[x, y] = a[x, y]]\)
Create Object

• Precondition: \( o \not\in O \)

• Primitive command: \texttt{create object} \( o \) \texttt{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

\textbf{command} \textit{cry} ⋅ \textit{havoc}(s_1 : u, s_2 : u, o_1 : v, o_2 : v, o_3 : w, o_4 : w)

create subject \( s_1 \) of type \( u \);
create object \( o_1 \) of type \( v \);
create object \( o_3 \) of type \( w \);
enter \( r \) into \( a[s_2, s_1] \);
enter \( r \) into \( a[s_2, o_2] \);
enter \( r \) into \( a[s_2, o_4] \)

end
Creation Graph

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

\[\text{command } \text{cry} \cdot \text{havoc}(s_1: u, s_2: u, o_1: v, o_3: w)\]
\begin{align*}
\quad & \text{create object } o_1 \text{ of type } v; \\
\quad & \text{create object } o_3 \text{ of type } w; \\
\quad & \text{enter } r \text{ into } a[s_2, s_1]; \\
\quad & \text{enter } r \text{ into } a[s_2, o_1]; \\
\quad & \text{enter } r \text{ into } a[s_2, o_3]
\end{align*}
\text{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 \textit{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”

• $\Sigma$ set of states
• $Q$ set of queries
• $e: \Sigma \times Q \rightarrow \{\text{true}, \text{false}\}$
  • Called entailment relation
• $T$ set of state transition rules
• $(\Sigma, Q, e, T)$ 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 $\sigma_0$ to $\sigma_1$ as $\sigma_0 \mapsto \sigma_1$
  
  • Emphasizing we’re looking at permissions

  • Multiple transitions are $\sigma_0 \mapsto^* \sigma_n$
    
    • $\Sigma_n$ said to be $\tau$-reachable from $\sigma_0$
Example: Take-Grant

• $\Sigma$ set of all possible protection graphs

• $Q$ set of queries

  \\{ can\textbullet share(\alpha, v_1, v_2, G_0) \mid \alpha \in R, v_1, v_2 \in G_0 \}\n
• $e(\sigma_0, q) = true$ if $q$ holds; $e(\sigma_0, q) = false$ if not

• $T$ set of sequences of take, grant, create, remove rules
Security Analysis Instance

• Let \((\Sigma, Q, e, T)\) be an access control scheme

• Tuple \((\sigma, q, \tau, \Pi)\) is security analysis instance, where:
  • \(\sigma \in \Sigma\) — \(\tau \in T\)
  • \(q \in Q\) — \(\Pi \) is \(\forall\) or \(\exists\)

• If \(\Pi\) is \(\exists\), existential security analysis
  • Is there a state \(\sigma'\) such that \(\sigma \mapsto^{*} \sigma', e(\sigma', q) = true\)?

• If \(\Pi\) is \(\forall\), universal security analysis
  • For all states \(\sigma'\) such that \(\sigma \mapsto^{*} \sigma'\), is \(e(\sigma', q) = true\)?
Example: Take-Grant

• $\sigma_0 = G_0$
• $q$ is can\(\cdot\)share($r, v_1, v_2, G_0$)
• $\tau$ is sequence of take-grant rules
• $\Pi$ is $\exists$
• Security analysis instance examines whether $v_1$ has $r$ rights over $v_2$ in graph with initial state $G_0$
• So safety question is security analysis instance