Foundational Results

Chapter 3
Overview

• Safety Question
• HRU Model
• Take-Grant Protection Model
• SPM, ESPM
  • Multiparent joint creation
• Expressive power
• Typed Access Matrix Model
• Comparing properties of models
What Is “Secure”? 

• Adding a generic right \( r \) where there was not one is “leaking”
  • In what follows, a right leaks if it was not present \textit{initially}
  • Alternately: not present \textit{in the previous state} (not discussed here)

• If a system \( S \), beginning in initial state \( s_0 \), cannot leak right \( r \), it is \textit{safe with respect to the right} \( r \)
  • Otherwise it is called \textit{unsafe with respect to the right} \( r \)
Safety Question

• Is there an algorithm for determining whether a protection system $S$ with initial state $s_0$ is safe with respect to a generic right $r$?
  • Here, “safe” = “secure” for an abstract model
Mono-Operational Commands

• Answer: yes

• Sketch of proof:
  Consider minimal sequence of commands \( c_1, \ldots, c_k \) to leak the right.
  • Can omit delete, destroy
  • Can merge all creates into one

Worst case: insert every right into every entry; with \( s \) subjects and \( o \) objects initially, and \( n \) rights, upper bound is \( k \leq n(s+1)(o+1) \)
General Case

• Answer: no

• Sketch of proof:
  Reduce halting problem to safety problem
  Turing Machine review:
  • Infinite tape in one direction
  • States $K$, symbols $M$; distinguished blank $b$
  • Transition function $\delta(k, m) = (k', m', L)$ means in state $k$, symbol $m$ on tape location replaced by symbol $m'$, head moves to left one square, and enters state $k'$
  • Halting state is $q_f$, TM halts when it enters this state
Mapping

Current state is $k$

<table>
<thead>
<tr>
<th></th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
</tr>
</thead>
<tbody>
<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>C</td>
<td>own</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_4$</td>
<td></td>
<td></td>
<td>D</td>
<td>end</td>
</tr>
</tbody>
</table>
Mapping

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

\begin{align*}
1 & \quad 2 & \quad 3 & \quad 4 \\
A & \quad B & \quad X & \quad D & \quad \ldots
\end{align*}

\begin{tabular}{|c|c|c|c|}
\hline
$s_1$ & $s_2$ & $s_3$ & $s_4$ \\
\hline
$s_1$ & A & & own & \\
$s_2$ & & B & own & \\
$s_3$ & & X & own & \\
$s_4$ & & & D & \textit{k}_1 \text{ end} \\
\hline
\end{tabular}
Command Mapping

• $\delta(k, C) = (k_1, X, R)$ at intermediate becomes

```
command $c_{k,C}(s_3,s_4)$
if own in $A[s_3,s_4]$ and $k$ in $A[s_3,s_3]$
    and $C$ in $A[s_3,s_3]$
then
    delete $k$ from $A[s_3,s_3]$;
    delete $C$ from $A[s_3,s_3]$;
    enter $X$ into $A[s_3,s_3]$;
    enter $k_1$ into $A[s_4,s_4]$;
end
```
Mapping

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>X</td>
<td>Y</td>
<td>...</td>
</tr>
</tbody>
</table>

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>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
<th>$s_5$</th>
</tr>
</thead>
<tbody>
<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></td>
<td>X</td>
<td>own</td>
<td></td>
</tr>
<tr>
<td>$s_4$</td>
<td></td>
<td></td>
<td>Y</td>
<td>own</td>
</tr>
<tr>
<td>$s_5$</td>
<td></td>
<td></td>
<td></td>
<td>b $k_2$ end</td>
</tr>
</tbody>
</table>
Command Mapping

• $\delta(k_1, D) = (k_2, Y, R)$ at end becomes

  \[
  \begin{align*}
  \text{command} & \; \text{crightmost}_{k,c}(s_4,s_5) \\
  \text{if} & \; \text{end in } A[s_4,s_4] \; \text{and} \; k_1 \; \text{in } A[s_4,s_4] \\
  & \; \text{and} \; D \; \text{in } A[s_4,s_4] \\
  \text{then} \\
  & \; \text{delete} \; \text{end from } A[s_4,s_4]; \\
  & \; \text{delete} \; k_1 \; \text{from } A[s_4,s_4]; \\
  & \; \text{delete} \; D \; \text{from } A[s_4,s_4]; \\
  & \; \text{enter} \; Y \; \text{into } A[s_4,s_4]; \\
  & \; \text{create subject } s_5; \\
  & \; \text{enter} \; \text{own into } A[s_4,s_5]; \\
  & \; \text{enter} \; \text{end into } A[s_5,s_5]; \\
  & \; \text{enter} \; k_2 \; \text{into } A[s_5,s_5]; \\
  \text{end}
  \end{align*}
  \]
Rest of Proof

• Protection system exactly simulates a TM
  • Exactly 1 *end* right in ACM
  • 1 right in entries corresponds to state
  • Thus, at most 1 applicable command

• If TM enters state $q_f$, then right has leaked

• If safety question decidable, then represent TM as above and determine if $q_f$ leaks
  • Implies halting problem decidable

• Conclusion: safety question undecidable
Other Results

• Set of unsafe systems is recursively enumerable
• Delete create primitive; then safety question is complete in P-SPACE
• Delete destroy, delete primitives; then safety question is undecidable
  • Systems are monotonic
• Safety question for biconditional protection systems is decidable
• Safety question for monoconditional, monotonic protection systems is decidable
• Safety question for monoconditional protection systems with create, enter, delete (and no destroy) is decidable.
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
System

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

\[ G \vdash_x G' \] apply a rewriting rule \( x \) (witness) to \( G \) to get \( G' \)

\[ G \vdash^* G' \] apply a sequence of rewriting rules (witness) to \( G \) to get \( G' \)

\( R = \{ t, g, r, w, ... \} \) set of rights
Rules

take

grant
More Rules

create

remove

These four rules are called the *de jure* rules
Symmetry

1. $x$ creates ($tg$ to new) $v$
2. $z$ takes ($g$ to $v$) from $x$
3. $z$ grants ($\alpha$ to $y$) to $v$
4. $x$ takes ($\alpha$ to $y$) from $v$

Similar result for grant
Islands

- \textit{tg}-path: path of distinct vertices connected by edges labeled \textit{t} or \textit{g}
  - Call them “\textit{tg}-connected”

- island: maximal \textit{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 \( \{ t^{*}g \} \cup \{ \nu \} \)
  • 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 \( \{ t^{*} \} \cup \{ \nu \} \)
  • Means \( x \) can acquire any rights \( y \) has
Bridges

• bridge: \( tg \)-path between subjects \( x, y \), with associated word in
  \[ \{ \overrightarrow{t^*}, \overrightarrow{t^*}, \overrightarrow{t^*g}, \overrightarrow{t^*g}, \overrightarrow{t^*} \} \]
  • rights can be transferred between the two endpoints
  • \textit{not} an island as intermediate vertices are objects
Example

- islands \{ p, u \} \{ w \} \{ y, s' \}
- bridges uvw; wxy
- initial span p (associated word v)
- terminal span s's (associated word t)
can\•share Predicate

Definition:

• \textit{can\•share}(r, x, y, G_0) if, and only if, there is a sequence of protection graphs \( G_0, \ldots, G_n \) such that \( G_0 \vdash^* G_n \) using only \textit{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
Example Interpretation

• ACM is generic
  • Can be applied in any situation
• Take-Grant has specific rules, rights
  • Can be applied in situations matching rules, rights
• Question: what states can evolve from a system that is modeled using the Take-Grant Model?
Take-Grant Generated Systems

• Theorem: $G_0$ protection graph with 1 vertex, no edges; $R$ set of rights. Then $G_0 \vdash^* G$ iff:
  • $G$ finite directed graph consisting of subjects, objects, edges
  • Edges labeled from nonempty subsets of $R$
  • At least one vertex in $G$ has no incoming edges
Outline of Proof

⇒: By construction; G final graph in theorem
- Let \( x_1, \ldots, x_n \) be subjects in \( G \)
- Let \( x_1 \) have no incoming edges
- Now construct \( G' \) as follows:
  1. Do “\( x_1 \) creates (\( \alpha \cup \{ g \} \) to) new subject \( x_i \)”
  2. For all (\( x_i, x_j \)) where \( x_i \) has a rights over \( x_j \), do
     “\( x_1 \) grants (\( \alpha \) to \( x_j \)) to \( x_i \)”
  3. Let \( \beta \) be rights \( x_i \) has over \( x_j \) in \( G \). Do
     “\( x_1 \) removes ((\( \alpha \cup \{ g \} \) – \( \beta \) to) \( x_j \)”
- Now \( G' \) is desired \( G \)
Outline of Proof

$: \text{Let } v \text{ be initial subject, and } G_0 \vdash^* G$

• Inspection of rules gives:
  • $G$ is finite
  • $G$ is a directed graph
  • Subjects and objects only
  • All edges labeled with nonempty subsets of $R$

• Limits of rules:
  • None allow vertices to be deleted so $v$ in $G$
  • None add incoming edges to vertices without incoming edges, so $v$ has no incoming edges
Example: Shared Buffer

- Goal: \( p, q \) to communicate through shared buffer \( b \) controlled by trusted entity \( s \)
  1. \( s \) creates (\( \{r, w\} \) to new object) \( b \)
  2. \( s \) grants (\( \{r, w\} \) to \( b \)) to \( p \)
  3. \( s \) grants (\( \{r, w\} \) to \( b \)) to \( q \)
can\textbullet steal Predicate

Definition:

\[\text{can\textbullet steal}(r, x, y, G_0)\] if, and only if, there is no edge from \(x\) to \(y\) labeled \(r\) in \(G_0\), and the following hold simultaneously:

- There is edge from \(x\) to \(y\) labeled \(r\) in \(G_n\)
- There is a sequence of rule applications \(\rho_1, \ldots, \rho_n\) such that \(G_{i-1} \vdash G_i\) using \(\rho_i\)
- For all vertices \(v, w\) in \(G_{i-1}\), if there is an edge from \(v\) to \(y\) in \(G_0\) labeled \(r\), then \(\rho_i\) is \textit{not} of the form \(v\) grants \((r\ to \ y)\) to \(w\)\]
Example

can\textit{steal}(α, s, w, G₀):

1. u grants (t to v) to s
2. s takes (t to u) from v
3. s takes (α to w) from u
can\textbullet steal Theorem

- \textit{can\textbullet steal}(r, x, y, G_0) if, and only if, the following hold simultaneously:
  a) There is no edge from x to y labeled r in G_0
  b) There exists a subject x' such that x' = x or x' initially spans to x
  c) There exists a vertex s with an edge labeled \alpha to y in G_0
  d) \textit{can\textbullet share}(t, x', s, G_0) holds
Outline of Proof

⇒: Assume conditions hold

• x subject
  • x gets t rights to s, then takes α to y from s

• x object
  • can•share(t, x', s, G₀) holds
  • If x' has no α edge to y in G₀, x' takes (α to y) from s and grants it to x
  • If x' has a edge to y in G₀, x' creates surrogate x'', gives it (t to s) and (g to x''); then x'' takes (α to y) and grants it to x
Outline of Proof

\[ \iff: \text{Assume } can\cdot\text{steal}(\alpha, x, y, G_0) \text{ holds} \]

- First two conditions immediate from definition of \textit{can\cdotsteal}, \textit{can\cdotshare}
- Third condition immediate from theorem of conditions for \textit{can\cdotshare}
- Fourth condition: \( \rho \) minimal length sequence of rule applications deriving \( G_n \) from \( G_0 \); \( i \) smallest index such that \( G_{i-1} \vdash G_i \) by rule \( \rho_i \) and adding \( \alpha \) from some \( p \) to \( y \) in \( G_i \)
  - What is \( \rho_i \)?
Outline of Proof

• Not remove or create rule
  • $y$ exists already

• Not grant rule
  • $G_i$ first graph in which edge labeled $\alpha$ to $y$ is added, so by definition of $can\cdot share$, cannot be grant

• take rule: so $can\cdot share(t, p, s, G_0)$ holds
  • So is subject $s'$ such that $s' = s$ or terminally spans to $s$
  • Sequence of islands with $x' \in I_1$ and $s' \in I_n$

• Derive witness to $can\cdot share(t, x', s, G_0)$ that does not use “$s$ grants ($\alpha$ to $y$) to” anyone
Conspiracy

• Minimum number of actors to generate a witness for $\text{can•share}(\alpha, x, y, G_0)$

• Access set describes the “reach” of a subject

• Deletion set is set of vertices that cannot be involved in a transfer of rights

• Build conspiracy graph to capture how rights flow, and derive actors from it
Example

```
x  t  a  g  b  g  c  t  d  g  e  r  z
y  t  f  g  h  g  i  g  j
```
Access Set

• Access set $A(y)$ with focus $y$: set of vertices:
  • $\{y\}$
  • $\{x \mid y \text{ initially spans to } x\}$
  • $\{x' \mid y \text{ terminally spans to } x\}$

• Idea is that focus can give rights to, or acquire rights from, a vertex in this set
Example

- $A(x) = \{ x, a \}$
- $A(b) = \{ b, a \}$
- $A(c) = \{ c, b, d \}$
- $A(d) = \{ d \}$
- $A(e) = \{ e, d, i, j \}$
- $A(f) = \{ f, y \}$
- $A(h) = \{ h, f, i \}$
- $A(y) = \{ y \}$
Deletion Set

- Deletion set $\delta(y, y')$: contains those vertices in $A(y) \cap A(y')$ such that:
  - $y$ initially spans to $z$ and $y'$ terminally spans to $z$;
  - $y$ terminally spans to $z$ and $y'$ initially spans to $z$;
  - $z = y$
  - $z = y'$

- Idea is that rights can be transferred between $y$ and $y'$ if this set non-empty.
Example

- $\delta(x, b) = \{ a \}$
- $\delta(b, c) = \{ b \}$
- $\delta(c, d) = \{ d \}$
- $\delta(c, e) = \{ d \}$
- $\delta(d, e) = \{ d \}$
- $\delta(y, f) = \{ y \}$
- $\delta(h, f) = \{ f \}$
Conspiracy Graph

• Abstracted graph $H$ from $G_0$:
  • Each subject $x \in G_0$ corresponds to a vertex $h(x) \in H$
  • If $\delta(x, y) \neq \emptyset$, there is an edge between $h(x)$ and $h(y)$ in $H$
• Idea is that if $h(x)$, $h(y)$ are connected in $H$, then rights can be transferred between $x$ and $y$ in $G_0$
Example
Results

• $I(x)$: $h(x)$, all vertices $h(y)$ such that $y$ initially spans to $x$
• $T(x)$: $h(x)$, all vertices $h(y)$ such that $y$ terminally spans to $x$
• Theorem: can•share($\alpha$, $x$, $y$, $G_0$) iff there exists a path from some $h(p)$ in $I(x)$ to some $h(q)$ in $T(y)$
• Theorem: $l$ vertices on shortest path between $h(p)$, $h(q)$ in above theorem; $l$ conspirators necessary and sufficient to witness
Example: Conspirators

\[ l(x) = \{ h(x) \}, \quad T(z) = \{ h(e) \} \]

- Path between \( h(x) \), \( h(e) \) so can\textit{share}(r, x, z, G_0)
- Shortest path between \( h(x) \), \( h(e) \) has 4 vertices

\[ \Rightarrow \text{Conspirators are e, c, b, x} \]
Example: Witness

1. e grants (r to z) to d
2. c takes (r to z) from d
3. c grants (r to z) to b
4. b grants (r to z) to a
5. x takes (r to z) from a
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: $\text{link}_i(X, Y)$ if $X$ can assert some control right over $Y$

• Conjunction of disjunction of:
  • $X/z \in \text{dom}(X)$
  • $X/z \in \text{dom}(Y)$
  • $Y/z \in \text{dom}(X)$
  • $Y/z \in \text{dom}(Y)$
  • true
Examples

• Take-Grant:
  \[ link(X, Y) = \frac{Y}{g} \in \text{dom}(X) \lor \frac{X}{t} \in \text{dom}(Y) \]

• Broadcast:
  \[ link(X, Y) = \frac{X}{b} \in \text{dom}(X) \]

• Pull:
  \[ link(X, Y) = \frac{Y}{p} \in \text{dom}(Y) \]
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
  • $TS = \{ \text{subjects} \}$, $TO = \{ \text{objects} \}$
  • $RC = \{ tc, gc \}$, $RI = \{ rc, wc \}$
  • $link(p, q) = p/t \in \text{dom}(q) \lor q/g \in \text{dom}(p)$
  • $f(\text{subject}, \text{subject}) = \{ \text{subject, object} \} \times \{ tc, gc, rc, wc \}$
Create Operation

- Must handle type, tickets of new entity
- Relation $cc(a, b)$ [$cc$ for can-create]
  - Subject of type $a$ can create entity of type $b$
- Rule of acyclic creates:

```
\[ \begin{array}{c}
a \quad \rightarrow \quad b \\
c \quad \rightarrow \quad d \\
\end{array} \quad \begin{array}{c}
a \quad \rightarrow \quad b \\
c \quad \rightarrow \quad d \\
\end{array} \]
```
Types

• \( cr(a, b) \): tickets created when subject of type \( a \) creates entity of type \( b \) [\( cr \) for \( create-rule \)]

• \( B \) object: \( cr(a, b) \subseteq \{ b/r:c \in RI \} \)
  • \( A \) gets \( B/r:c \) iff \( b/r:c \in cr(a, b) \)

• \( B \) subject: \( cr(a, b) \) has two subsets
  • \( cr_p(a, b) \) 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: Owner-Based Policy

- Users can create files, creator can give itself any inert rights over file
  - $cc = \{ (\text{user}, \text{file}) \}$
  - $cr(\text{user}, \text{file}) = \{ \text{file}/r:c \mid r \in RI \}$
- Attenuating, as graph is acyclic, loop free
Example: Take-Grant

- Say subjects create subjects (type s), objects (type o), but get only inert rights over latter
  - \( cc = \{ (s, s), (s, o) \} \)
  - \( cr_c(a, b) = \emptyset \)
  - \( cr_p(s, s) = \{s/tc, s/gc, s/rc, s/wc\} \)
  - \( cr_p(s, o) = \{s/rc, s/wc\} \)
- Not attenuating, as no self tickets provided; subject creates subject
Safety Analysis

• Goal: identify types of policies with tractable safety analyses
• Approach: derive a state in which additional entries, rights do not affect the analysis; then analyze this state
  • Called a *maximal state*
Definitions

- System begins at initial state
- Authorized operation causes legal transition
- Sequence of legal transitions moves system into final state
  - This sequence is a history
  - Final state is derivable from history, initial state
More Definitions

• States represented by \( h \)
• Set of subjects \( SUB^h \), entities \( ENT^h \)
• Link relation in context of state \( h \) is \( link^h \)
• Dom relation in context of state \( h \) is \( dom^h \)
$path^h(X,Y)$

- $X$, $Y$ connected by one link or a sequence of links
- Formally, either of these hold:
  - for some $i$, $link_i^h(X, Y)$; or
  - there is a sequence of subjects $X_0, \ldots, X_n$ such that $link_i^h(X, X_0)$, $link_i^h(X_n, Y)$, and for $k = 1, \ldots, n$, $link_i^h(X_{k-1}, X_k)$
- If multiple such paths, refer to $path_j^h(X, Y)$
Capacity $cap(path^h(X,Y))$

• Set of tickets that can flow over $path^h(X,Y)$
  • If $link_i^h(X,Y)$: set of tickets that can be copied over the link (i.e., $f_i(\tau(X), \tau(Y)))$
  • Otherwise, set of tickets that can be copied over all links in the sequence of links making up the $path^h(X,Y)$

• Note: all tickets (except those for the final link) must be copyable
Flow Function

• Idea: capture flow of tickets around a given state of the system
• Let there be $m$ $path^h$s between subjects $X$ and $Y$ in state $h$. Then flow function

$$flow^h: SUB^h \times SUB^h \rightarrow 2^{T \times R}$$

is:

$$flow^h(X,Y) = \bigcup_{i=1,...,m} cap(path^i_{h}(X,Y))$$
Properties of Maximal State

• Maximizes flow between all pairs of subjects
  • State is called *
  • Ticket in $\text{flow}^*(X,Y)$ means there exists a sequence of operations that can copy the ticket from $X$ to $Y$

• Questions
  • Is maximal state unique?
  • Does every system have one?
Formal Definition

• Definition: $g \leq_0 h$ holds iff for all $X, Y \in SUB^0$, $\text{flow}^g(X,Y) \subseteq \text{flow}^h(X,Y)$.
  • Note: if $g \leq_0 h$ and $h \leq_0 g$, then $g$, $h$ equivalent
  • Defines set of equivalence classes on set of derivable states

• Definition: for a given system, state $m$ is maximal iff $h \leq_0 m$ for every derivable state $h$

• Intuition: flow function contains all tickets that can be transferred from one subject to another
  • All maximal states in same equivalence class
Maximal States

• Lemma. Given arbitrary finite set of states $H$, there exists a derivable state $m$ such that for all $h \in H$, $h \leq_0 m$

• Outline of proof: induction
  • Basis: $H = \emptyset$; trivially true
  • Step: $|H'| = n + 1$, where $H' = G \cup \{h\}$. By IH, there is a $g \in G$ such that $x \leq_0 g$ for all $x \in G$. 
Outline of Proof

• M interleaving histories of $g$, $h$ which:
  • Preserves relative order of transitions in $g$, $h$
  • Omits second create operation if duplicated

• $M$ ends up at state $m$

• If $path^g(X,Y)$ for $X, Y \in SUB^g$, $path^m(X,Y)$
  • So $g \leq_0 m$

• If $path^h(X,Y)$ for $X, Y \in SUB^h$, $path^m(X,Y)$
  • So $h \leq_0 m$

• Hence $m$ maximal state in $H'$
Answer to Second Question

• Theorem: every system has a maximal state *

• Outline of proof: \( K \) is set of derivable states containing exactly one state from each equivalence class of derivable states
  
  • Consider \( X, Y \) in \( SUB^0 \). Flow function’s range is \( 2^{T \times R} \), so can take at most \( 2^{|T \times R|} \) values. As there are \( |SUB^0|^2 \) pairs of subjects in \( SUB^0 \), at most \( 2^{|T \times R|} \cdot |SUB^0|^2 \) distinct equivalence classes; so \( K \) is finite

• Result follows from lemma
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 $X/r:c$ to be in $\text{dom}^h(Y)$:
  • $X/r:c \in \text{dom}^0(Y)$ means $X, Y \in \text{ENT}^0$, so trivially $\sigma(X)/r:c \in \text{dom}^g(\sigma(Y))$ holds
  • A create added $X/r:c \in \text{dom}^h(Y)$: previous lemma says $\sigma(X)/r:c \in \text{dom}^g(\sigma(Y))$ holds
  • A demand added $X/r:c \in \text{dom}^h(Y)$: corresponding demand operation in $G$ gives $\sigma(X)/r:c \in \text{dom}^g(\sigma(Y))$
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. $link^h_i(Y, Z)$
  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 $link^g_i(\sigma(Y), \sigma(Z))$

• 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 $link^h_i(X, Y)$, then $link^g_i(\sigma(X), \sigma(Y))$
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

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

• 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 $\text{path}^h(X, Y) = 1$. By definition of $\text{path}^h$, $\text{link}^h_i(X, Y)$, hence $\text{link}^g_i(\sigma(X), \sigma(Y))$. As $\sigma$ preserves type, this means

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

• Now assume this is true when $\text{path}^h(X, Y)$ 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 $\text{link}^h_j(Z, Y)$.

• By IH, there is a $\text{path}^g(\sigma(X), \sigma(Z))$ with same capacity as $\text{path}^h(X, Z)$

• By corollary, $\text{link}^g_j(\sigma(Z), \sigma(Y))$

• 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 ν be #u
  • Deriving history has no creates
  • By theorem,
    \[(\forall X,Y \in SUB^h)[flow^h(X, Y) \subseteq flow^#u(\sigma(X), \sigma(Y))]\]
  • If X ∈ SUB^0, \(\sigma(X) = X\), so:
    \[(\forall X,Y \in SUB^0)[flow^h(X, Y) \subseteq flow^#u(X, Y)]\]
• So #u is maximal state for system with acyclic attenuating scheme
  • #u derivable from u in time polynomial to |SUB^u|
  • Worst case computation for flow^#u is exponential in |TS|
Safety Result

• If the scheme is acyclic and attenuating, the safety question is decidable
Expressive Power

• How do the sets of systems that models can describe compare?
  • If HRU equivalent to SPM, SPM provides more specific answer to safety question
  • If HRU describes more systems, SPM applies only to the systems it can describe
HRU vs. SPM

- SPM more abstract
  - Analyses focus on limits of model, not details of representation
- HRU allows revocation
  - SMP has no equivalent to delete, destroy
- HRU allows multiparent creates
  - SMP cannot express multiparent creates easily, and not at all if the parents are of different types because can\textit{\textbullet}create allows for only one type of creator
Multiparent Create

- Solves mutual suspicion problem
  - Create proxy jointly, each gives it needed rights
- In HRU:

  \[
  \text{command } \text{multicreate}(s_0, s_1, o) \\
  \text{if } r \text{ in } a[s_0, s1] \text{ and } r \text{ in } a[s_1, s_0] \\
  \text{then} \\
  \quad \text{create object } o; \\
  \quad \text{enter } r \text{ into } a[s_0, o]; \\
  \quad \text{enter } r \text{ into } a[s_1, o]; \\
  \text{end}
  \]
SPM and Multiparent Create

• *cc* extended in obvious way
  • $cc \subseteq TS \times ... \times TS \times T$

• Symbols
  • $X_1, ..., X_n$ parents, $Y$ created
  • $R_{1,i}, R_{2,i}, R_{3}, R_{4,i} \subseteq R$

• Rules
  • $cr_{p,i}(\tau(X_1), ..., \tau(X_n)) = Y/R_{1,1} \cup X_i/R_{2,i}$
  • $cr_{C}(\tau(X_1), ..., \tau(X_n)) = Y/R_{3} \cup X_1/R_{4,1} \cup ... \cup X_n/R_{4,n}$
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\(\cdot\)create

- Following added to \(\text{can}\cdot\text{create}\):
  - \(\text{cc}(p_1) = a_1\)
  - \(\text{cc}(p_2, a_1) = a_2\)
  - \(\text{cc}(p_3, a_2) = a_3\)
    - 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
  • \( cr_{Pfirst}(p_2, a_1, a_2) = \emptyset \)
  • \( cr_{Psecond}(p_2, a_1, a_2) = \emptyset \)
  • \( cr_C(p_2, a_1, a_2) = p_2/Rtc \cup a_1/tc \)
    • Agent’s parent set to creating parent and agent; agent has all rights over parent (but not over agent)
Creation Rules

- $cr_{P_{\text{first}}}(p_3, a_2, a_3) = \emptyset$
- $cr_{P_{\text{second}}}(p_3, a_2, a_3) = \emptyset$
- $cr_C(p_3, a_2, a_3) = \frac{p_3}{Rtc} \cup \frac{a_2}{tc}$
  - Agent’s parent set to creating parent and agent; agent has all rights over parent (but not over agent)
- $cr_p(a_3, s) = \emptyset$
- $cr_C(a_3, s) = \frac{a_3}{tc}$
  - 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

- \( f_1(a_2, a_1) = \frac{a_1}{t} + \frac{c}{Rtc} \)
- \( f_1(a_3, a_2) = \frac{a_2}{t} + \frac{c}{Rtc} \)
- \( f_2(s, a_3) = \frac{a_3}{t} + \frac{c}{Rtc} \)
- \( f_3(a_1, c) = \frac{p_1}{R_{4,1}} \)
- \( f_3(a_2, c) = \frac{p_2}{R_{4,2}} \)
- \( f_3(a_3, c) = \frac{p_3}{R_{4,3}} \)
- \( f_4(a_1, p_1) = \frac{c}{R_{1,1}} + \frac{p_1}{R_{2,1}} \)
- \( f_4(a_2, p_2) = \frac{c}{R_{1,2}} + \frac{p_2}{R_{2,2}} \)
- \( f_4(a_3, p_3) = \frac{c}{R_{1,3}} + \frac{p_3}{R_{2,3}} \)
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 $C/R_3t$
Construction

• Only $\text{link}_2(S, A_3)$ true $\Rightarrow$ apply $f_2$
  • $A_3$ has $P_3/Rtc \cup A_2/t \cup A_3/t \cup C/Rtc$

• Now $\text{link}_1(A_3, A_2)$ true $\Rightarrow$ apply $f_1$
  • $A_2$ has $P_2/Rtc \cup A_1/tc \cup A_2/t \cup C/Rtc$

• Now $\text{link}_1(A_2, A_1)$ true $\Rightarrow$ apply $f_1$
  • $A_1$ has $P_2/Rtc \cup A_1/t \cup C/Rtc$

• Now all $\text{link}_3$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
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 $e'$

![Diagram showing nodes A1, A2, A3, and edges P1, P2, P3]
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
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

- **W, X₁, X₂** nodes in **B**
  - **W** creates **Y₁, Y₂, Y₃** using single parent create rule, and adds edges for **X₁, X₂** to all using edge adding rule
  - **Y₁** creates **Z**, again using single parent create rule; now must add edge from **Y₂** to **Z** to simulate **A**
  - Use same edge adding rule to add edge from **Y₃** to **Z**: cannot duplicate this in scheme **A**!
Meaning

• Scheme $B$ cannot simulate scheme $A$, contradicting hypothesis
• ESPM more expressive than SPM
  • ESPM multiparent and monotonic
  • SPM monotonic but single parent
Typed Access Matrix Model

• Like ACM, but with set of types $T$
  • All subjects, objects have types
  • Set of types for subjects $TS$

• Protection state is $(S, O, \tau, A)$
  • $\tau : O \rightarrow T$ 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: 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: $o \notin O$

• Primitive command: **create object $o$ of type $t$**

• Postconditions:
  • $S' = S$, $O' = O \cup \{ o \}$
  • $(\forall y \in O)[\tau'(y) = \tau(y)]$, $\tau'(o) = t$
  • $(\forall x \in S')[a'[x, o] = \emptyset]$
  • $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$
Definitions

• MTAM Model: TAM model without **delete, destroy**
  • MTAM is Monotonic TAM

• $\alpha(x_1:t_1, ..., x_n:t_n)$ 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

\[ \text{command } cry\cdot havoc(s_1 : u, s_2 : u, o_1 : v, o_2 : v, \]
\[ \quad o_3 : w, o_4 : w) \]
\[ \text{create subject } s_1 \text{ of type } u; \]
\[ \text{create object } o_1 \text{ of type } v; \]
\[ \text{create object } o_3 \text{ of type } w; \]
\[ \text{enter } r \text{ into } a[s_2, s_1]; \]
\[ \text{enter } r \text{ into } a[s_2, o_2]; \]
\[ \text{enter } r \text{ into } a[s_2, o_4] \]
\[ \text{end} \]
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
command cry•havoc($s_1 : u$, $s_2 : u$, $o_1 : v$, $o_3 : w$)

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 $a[s_2, o_1]$;
enter $r$ into $a[s_2, o_3]$
end
Creation Graph

- $v$, $w$ child types
- $u$ parent type
- Graph: lines from parent types to child types
- This one has no cycles
Theorems

• Safety decidable for systems with acyclic MTAM schemes
  • In fact, it’s \(NP\)-hard

• Safety for acyclic ternary MATM decidable in time polynomial in the size of initial ACM
  • “Ternary” means commands have no more than 3 parameters
  • Equivalent in expressive power to MTAM
Security Properties

• Question: given two models, do they have the same security properties?
  • First comes theory
  • Then comes an example comparison

• Basic idea: view access request as query asking if subject has right to perform action on object
Alternate Definition of “Scheme”

- $\Sigma$ set of states
- $Q$ set of queries
- $e: \Sigma \times Q \rightarrow \{\text{true, false}\}$
  - 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

• Σ set of all possible protection graphs
• Q set of queries
  \{ can•share(α, v_1, v_2, G_0) \mid α ∈ R, v_1, v_2 ∈ G_0 \}
• e(σ_0, q) = true if q holds; e(σ_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 \(\sigma \mapsto_{\tau}^{*} \sigma'\), \(e(\sigma', q) = true\)?

• If \(\Pi\) is \(\forall\), universal security analysis
  • For all states \(\sigma'\) such that \(\sigma \mapsto_{\tau}^{*} \sigma'\), 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
Comparing Two Models

• Each query in $A$ corresponds to a query in $B$
• Each (state, state transition) in $A$ corresponds to (state, state transition) in $B$

Formally:

• $A = (\Sigma^A, Q^A, e^A, T^A)$ and $B = (\Sigma^B, Q^B, e^B, T^B)$
• mapping from $A$ to $B$ is:
  • $f : (\Sigma^A \times T^A) \cup Q^A \rightarrow (\Sigma^B \times T^B) \cup Q^B$
Image of Instance

• \( f \) mapping from \( A \) to \( B \)

• *image of a security analysis instance* 
  
  \((\sigma^A, q^A, \tau^A, \Pi)\) under \( f \) is \((\sigma^B, q^B, \tau^B, \Pi)\),

  where:

  • \( f((\sigma^A, \tau^A)) = (\sigma^B, \tau^B) \)
  • \( f(q^A) = q^B \)

• \( f \) is *security-preserving* if every security analysis instance in \( A \) is true iff its image is true
Composition of Queries

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

• Tuple \((\sigma, \phi, \tau, \Pi)\) is compositional security analysis instance, where \(\phi\) is propositional logic formula of queries from \(Q\)

• *image of compositional security analysis instance* defined similarly to previous

• \(f\) is *strongly security-preserving* if every compositional security analysis instance in \(A\) is true iff its image is true
State-Matching Reduction

• \( A = (\Sigma^A, Q^A, e^A, T^A) \), \( B = (\Sigma^B, Q^B, e^B, T^B) \), \( f \) mapping from \( A \) to \( B \)
• \( \sigma^A, \sigma^B \) equivalent under the mapping \( f \) when
  • \( e^A(\sigma^A, q^A) = e^B(\sigma^B, q^B) \)
• \( f \) state-matching reduction if for all \( \sigma^A \in S^A \), \( \tau^A \in T^A \),
  \( (\sigma^B, \tau^B) = f((\sigma^A, \tau^A)) \) has the following properties:
Property 1

• For every state $\sigma^A$ in scheme $A$ such that $\sigma^A \xrightarrow{\tau^*} \sigma^A$, there is a state $\sigma^B$ in scheme $B$ such that $\sigma^B \xrightarrow{\tau^*} \sigma^B$, and $\sigma^A$ and $\sigma^B$ are equivalent under the mapping $f$
  
  • That is, for every reachable state in $A$, a matching state in $B$ gives the same answer for every query
Property 2

• For every state $\sigma^B$ in scheme $B$ such that $\sigma^B \xrightarrow{\tau}^* \sigma'^B$, there is a state $\sigma'^A$ in scheme $A$ such that $\sigma^A \xrightarrow{\tau}^* \sigma'^A$, and $\sigma'^A$ and $\sigma'^B$ are equivalent under the mapping $f$
  • That is, for every reachable state in $B$, a matching state in $A$ gives the same answer for every query
Theorem

Mapping $f$ from scheme $A$ to $B$ is strongly security-preserving iff $f$ is a state-matching reduction
Proof ($\equiv$)

- Must show $(\sigma^A, \phi^A, \tau^A, \Pi) \text{ true iff } (\sigma^B, \phi^B, \tau^B, \Pi) \text{ true}$
- $\Pi$ is $\exists$: assume $\tau^A$-reachable state $\sigma'^A$ from $\sigma^A$ in which $\phi^A$ true
  - By property 1, there is a state $\sigma'^B$ corresponding to $\sigma'^A$ in which $\phi^B$ holds
- $\Pi$ is $\forall$: assume $\tau^A$-reachable state $\sigma'^A$ from $\sigma^A$ in which $\phi^A$ false
  - By property 1, there is a state $\sigma'^B$ corresponding to $\sigma'^A$ in which $\phi^B$ false
- Same for $\phi^B$ with $\tau^B$-reachable state $\sigma'^B$ from $\sigma^B$
- So $(\sigma^A, \phi^A, \tau^A, \Pi) \text{ true iff } (\sigma^B, \phi^B, \tau^B, \Pi) \text{ true}
Proof (⇐)

• Let \( f \) be a map from \( A \) to \( B \) but not state-matching reduction. Then there are \( \sigma^A \in S^A, \tau^A \in T^A, (\sigma^B, \tau^B) = f((\sigma^A, \tau^A)) \) violating at least one of the properties.

• Assume it’s property 1; \( \sigma^A, \sigma^B \) corresponding states. There is a \( \tau^A \)-reachable state \( \sigma^A' \) from \( \sigma^A \) such that no \( \tau^B \)-reachable state from \( \sigma^B \) is equivalent to \( \sigma^A' \).

• Generate \( \varphi^A \) and \( \varphi^B \) such that the existential compositional security analysis in \( A \) is true but in \( B \) is false.
  • To do this, look at each \( q^A \in Q^A \)
  • If \( e(\sigma^A', q^A) = true \), conjoin \( q^A \) to \( \varphi^A \); otherwise, conjoin \( \neg q^A \) to \( \varphi^A \)
  • Then \( e(\sigma^A', q^A) = true \) but for \( \varphi^B = f(\varphi^A) \) and all states \( \sigma^B' \) that are \( \tau^B \)-reachable from \( \sigma^B \), \( e(\sigma^B', q^B) = false \)

• Thus, \( f \) is not strongly security-preserving.

• Argument for property 2 is similar.
Expressive Power

If access control model $MA$ has a scheme that cannot be mapped into a scheme in access control model $MB$ using a state-matching reduction, then model $MB$ is \textit{less expressive than} model $MA$.

If every scheme in model $MA$ can be mapped into a scheme in model $MB$ using a state-matching reduction, then model $MB$ is \textit{as expressive as} model $MA$.

If $MA$ is as expressive as $MB$, and $MB$ is as expressive as $MA$, the models are \textit{equivalent}

\begin{itemize}
  \item Note this does not assume monotonicity, unlike earlier definition
\end{itemize}
Augmented Typed Access Control Matrix

• Add a test for the absence of rights to TAM

```plaintext
command add\cdot right(s\!:\!u, o\!:\!v)
    if own in a[s,o] and r not in a[s,o]
    then
        enter r into a[s,o]
    end
```

• How does this affect the answer to the safety question?
Safety Question

• ATAM can be mapped onto TAM
• But will the mapping, or any such mapping, preserve security properties?
• Approach: consider TAM as an access control model
TAM as Access Control Model

• $S$ set of subjects; $S_\sigma$ subjects in state $\sigma$
• $O$ set of objects; $O_\sigma$ objects in state $\sigma$
• $R$ set of rights; $R_\sigma$ rights in state $\sigma$
• $T$ set of types; $T_\sigma$ subjects in state $\sigma$
• $t : S_\sigma \cup O_\sigma \rightarrow T_\sigma$ gives type of any subject or object
• State $\sigma$ defined as $(S_\sigma, O_\sigma, R_\sigma, T_\sigma, t)$
• In TAM, query is of form “is $r \in a[s,o]$”, and $e(s, r \in a[s,o])$ true iff $s \in S_\sigma, o \in O_\sigma, r \in R_\sigma, r \in a_\sigma[s,o]$ are true
ATAM as Access Control Model

Same as TAM with one addition:

• ATAM also allows queries of form “is $r \notin a[s,o]$”, and $e(s, r \notin a[s,o])$
  true iff $s \in S_\sigma$, $o \in O_\sigma$, $r \in R_\sigma$, $r \notin a_\sigma[s,o]$ are true
Theorem

A state-matching reduction from ATAM to Tam does not exist.

Outline of proof: by contradiction

• Consider two state transitions, one that creates subject and one that adds right $r$ to an element of the matrix

• Can determine an upper bound on the number of answers to TAM query a command can change; depends on state and commands
Proof

• Assume $f$ is state-matching reduction from ATAM to TAM

• Consider simple ATAM scheme:
  • Initial state $\sigma_0$ has no subjects, objects
  • All entities have type $t$
  • Only one right $r$
  • Query $q_{ij} = r \in a[s,o]$; query $q_{ij} = r \notin a[s,o]$
  • 2 state transition rules
    • $make\cdotsubj(s : t)$ creates subject $s$ of type $t$
    • $add\cdotright(x : t, y : t)$ adds right $r$ to $a[x, y]$
Proof

• TAM: superscript $T$ represents components of that system
  • So initial state is $\sigma_0^T = f(\sigma_0)$, transitions are $\tau^T = f(\tau)$
• By definition of state-matching reduction, how $f$ maps queries does not depend on initial state or state transitions of a model
• Let $p, q$ be queries in ATAM and $p^T, q^T$ the corresponding queries in TAM; if $p \neq q$, then $p^T \neq q^T$
• As commands in TAM execute, they can change the value (response) of $q_{ij}$
• Upper bound on the number of values of queries a single command can change is $m$ (number of $\text{enter}$ or $\text{add\_right}$ operations)
Proof

- Choose $n > m$

- In ATAM, construct state $\sigma_k$ such that:
  - $\sigma_0 \rightarrow^* \sigma_k$; and
  - $e(\sigma_k, \neg q_{1,1} \land q_{1,1} \land \ldots \land \neg q_{n,n} \land q_{n,n})$ is true

- So $e(\sigma_k, q_{i,j})$ is false, $e(\sigma_k, q_{i,j})$ is true for all $1 \leq i, j \leq n$

- As $f$ is a state-matching reduction, there is a state $\sigma_k^T$ in TAM that causes the corresponding queries to be answered the same way

- Consider $\sigma_0^T \rightarrow \sigma_1^T \rightarrow \ldots \rightarrow \sigma_k^T$; choose first state $\sigma_C^T$ such that $e(\sigma_C^T, q_{i,j}^T \lor q_{i,j}^T)$ is true for all $1 \leq i, j \leq n$
Proof

• In $\sigma_{C-1}^T$, $e(\sigma_{C-1}^T, q_v, w^T \lor \overline{q_v, w^T})$ is false for some $1 \leq v, w \leq n$, so $e(\sigma_{C-1}^T, \neg q_v, w^T \land \neg \overline{q_v, w^T})$ is true

• State $\sigma$ in ATAM for which $e(\sigma, \neg q_v, w \land \neg \overline{q_v, w})$ is true is one in which either $s_v$ or $s_w$ or both does not exist

• Thus in that state, one of the following 2 queries holds:
  • $Q_1 = \neg q_{v,1} \land \neg \overline{q_{v,1}} \land \ldots \land \neg q_{n,v} \land \neg \overline{q_{n,v}}$
  • $Q_1 = \neg q_{w,1} \land \neg \overline{q_{w,1}} \land \ldots \land \neg q_{n,w} \land \neg \overline{q_{n,w}}$

• So in TAM, $e(\sigma_{C-1}^T, Q_1^T \land Q_2^T)$ is true
Proof

• Now consider the transition from $\sigma_{C-1}T$ to $\sigma_C^T$
• Values of at least $n$ queries in $Q_1$ or $Q_2$ must change from false to true
• But each command can change at most $m < n$ queries
• This is a contradiction
• So no such $f$ can exist, proving the result

Thus, ATAM can express security properties that TAM cannot
Key Points

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