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 Iᵢ to Iᵢ₊₁
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(\( \alpha, x, y, G_0 \))**

- \( \text{Can}_\text{share}(\alpha, x, y, G_0) \) iff there is an \( \alpha \) edge from \( x \) to \( y \) in \( G_0 \) if:
  - \( \exists \) a vertex \( s \in G_0 \) with an \( s \) to \( y \) \( \alpha \) edge,
  - \( \exists \) a subject \( x' \) such that \( x' \rightarrow \cdots \rightarrow x \) \( initially spans \) to \( x \),
  - \( \exists \) a subject \( s' \) such that \( s' \rightarrow \cdots \rightarrow s \) \( terminally spans \) to \( s \), and
  - \( \exists \) islands \( I_1, \ldots, I_n \) such that \( x' \in I_1, s' \in I_n \) and there is a bridge from \( I_j \) to \( I_{j+1} \)
Theorem: Can_share(α, x, y, G₀)

- Corollary: There is an O(|V|+|E|) algorithm to test can_share: Decidable in linear time!

Theorem:
- Let G₀ contain exactly one vertex and no edges,
- R a set of rights.
- G₀ ⊨* 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 G₀ ⊨* 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

If part: G meets the requirement
- Assume v is the vertex with no incoming edge and apply rules
  1. Perform “v creates (α ∪ {g} to) new x_i” for all 2≤i ≤ n, and α is union of all labels on the incoming edges going into x_i in G
  2. For all pairs x, y with x α over y in G, perform “v grants (α to y) to x”
  3. If β is the set of rights x has over y in G, perform “v removes (α ∪ {g} - β) 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 $\langle r, w \rangle$ to new object $b$
- $s$ grants $\langle r, w \rangle$, $b$ to $p$
- $s$ grants $\langle r, w \rangle$, $b$ to $q$
Theft in Take-Grant Model

- \(\text{Can\_steal}(\alpha, x, y, G_0)\) is true if there is no \(\alpha\) edge from \(x\) to \(y\) in \(G_0\) and \(\exists\) sequence \(G_1, ..., G_n\) s. t.:
  - \(\exists\) \(\alpha\) edge from \(x\) to \(y\) in \(G_n\)
  - \(\exists\) rules \(\rho_1, ..., \rho_n\) that take \(G_{i-1} \vdash \rho_i G_i\) and
  - \(\forall v, w \in G_n, 1 \leq i < n, \text{if } \exists \alpha\text{ edge from }v\text{ to }y\text{ in }G_0\text{ then }\rho_i\text{ is not }v\text{ grants }\alpha\text{ to }y\text{ to }w\)

- Disallows owners of \(\alpha\) 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 (\(\alpha\) to w) from u

\(s\)
\(-\bullet\)
\(u\)
\(-\bullet\)
\(t\)
\(f\)
\(g\)
\(v\)
\(w\)
\(\alpha\)
Theorem: When Theft Possible

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

\begin{itemize}
    \item There is no \alpha edge from x to y in G_0,
    \item \exists subject \textit{x'} such that \textit{x'} = x or \textit{x'} initially spans to x, and
    \item \exists s with \alpha edge to y in G_0 and \textbf{can\_share}(t, x, s, G_0)
\end{itemize}

 proof:

\begin{itemize}
    \item \Rightarrow: Assume the three conditions hold
        \begin{itemize}
            \item x can get t right over s (x is a subject) and then take \alpha right over y from s
            \item \textit{x'} creates a surrogate to pass \alpha to x (x is an object)
        \end{itemize}
\end{itemize}

\textbf{Proof:}

\begin{itemize}
    \item Assume \textbf{can\_steal} is true:
        \begin{itemize}
            \item No \alpha edge from definition 3.10 in G_0,
            \item \textbf{Can\_share}(\alpha, x, y, G_0) from definition 3.10 condition (a): \alpha from x to y in G_n,
            \item s exists from \textbf{can\_share} and earlier theorem
            \item Show \textbf{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
        \end{itemize}
\end{itemize}
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 $z$ $\cup$
- $y$ terminally spans to $z$ and $y'$ initially spans to $z$ $\cup$
- $z = y \cup z = y'$

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 $y$ to $y'$
Theorem: \( \text{Can}_{\text{share}}(a, x, y, G_\theta) \) iff conspiracy path from an item in an island containing \( x \) to an item that can steal from \( y \).

- Conspirators required is shortest path in conspiracy graph.
- Example from book.
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**

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

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

Let $\delta(k, D) = (k_2, Y, L)$
where $k_2$ is the next state

---

**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 $1 \leq i < k$
Mapping

Command Mapping
(Left move)

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

command \( c_{kC}(s_l, s_{l-1}) \)
if own in \( a[s_{l-1}, s_l] \) and \( k \) in \( a[s_l, s_j] \) and \( C \) in \( a[s_j, s_i] \)
then
- delete \( k \) from \( A[s_p, s_q] \);
- delete \( C \) from \( A[s_p, s_q] \);
- enter \( X \) into \( A[s_p, s_q] \);
- enter \( k_1 \) into \( A[s_{l-1}, s_{l-1}] \);
end
After $\delta(k, C) = (k_1, X, L)$ where $k$ is the current state and $k_1$ the next state.

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

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

\[
\text{command } c_{k_i C}(s_i, s_{i+1}) \\
\text{if } \text{own in } a[s_i, s_{i+1}] \text{ and } k \text{ in } a[s_i, s_i] \text{ and } C \text{ in } a[s_i, s_i] \\
\text{then} \\
\quad \text{delete } k \text{ from } A[s_i, s_i]; \\
\quad \text{delete } C \text{ from } A[s_i, s_i]; \\
\quad \text{enter } X \text{ into } A[s_i, s_i]; \\
\quad \text{enter } k_1 \text{ into } A[s_i, s_{i+1}]; \\
\text{end}
\]
**Command Mapping**
*(Rightmost move)*

\[ \delta(k_1, D) = (k_2, Y, R) \] at end becomes

**command** rightmost \( k_C(s_{i+1}, s_{i+1}) \)

if end in \( a[s_i, s_i] \) and \( k_1 \) in \( a[s_i, s_i] \) and \( D \) in \( a[s_i, s_i] \)

then

- delete end from \( a[s_i, s_i] \);
- create subject \( s_{i+1} \);
- enter own into \( a[s_i, s_{i+1}] \);
- enter end into \( a[s_{i+1}, s_{i+1}] \);
- delete \( k_1 \) from \( a[s_i, s_i] \);
- delete \( D \) from \( a[s_i, s_i] \);

**Mapping**

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

\[ \begin{array}{|c|c|c|c|c|}
\hline
1 & 2 & 3 & 4_i & \text{head} \\
\hline
A & B & X & Y \\
\hline
s_1 & s_2 & s_3 & s_4 & s_5 \\
\hline
\end{array} \]

\[ \begin{array}{|c|c|c|}
\hline
s_1 & s_2 & s_3 \\
\hline
A & own & \\
\hline
B & own & \\
\hline
X & own & \\
\hline
Y & own & \\
\hline
b & k_2 & end \\
\hline
\end{array} \]
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 $q_f$, then right has leaked
- If safety question decidable, then represent TM as above and determine if $q_f$ 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
  - $\tau(X)$ is the type of entity $X$
- A ticket describes a right
  - Consists of an entity name and a right symbol: $X/r$
    - Possessor of the ticket $X/r$ has right $r$ over entity $X$
    - $Y$ has tickets $X/r, X/w \Rightarrow Y$ has tickets $X/w$
  - Each entity $X$ has a set $dom(X)$ of tickets $Y/r$
  - $\tau(X/rc) = \tau(X)/rc$ 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
    - $X/z \in \text{dom}(X)$; $X/z \in \text{dom}(Y)$
    - $Y/z \in \text{dom}(X)$; $Y/z \in \text{dom}(Y)$
    - 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
  - \( f: TS \times TS \rightarrow 2^{TS} \) (range is copyable rights)
- \( X/r:c \) can be copied from \( dom(Y) \) to \( dom(Z) \) iff \( \exists i \) 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 \( F \) iff \( U \) 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 possess the right to do so (copy)
  - \( f(user, user) = \{ file/r, file/w, file/a, file/x \} \)
    - Can copy read, write, append, execute
SPM Example 1

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

Therefore, Peter can give ticket $Doom/xc$ 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
  - $\text{link}(p, q) = p/t \in \text{dom}(q) \lor q/t \in \text{dom}(p)$
  - $f(\text{subject}, \text{subject}) = \{ \text{subject}, \text{object} \} \times \{ tc, gc, rc, wc \}$
    - Note that any rights can be transferred in T-G model
Demand

- A subject can demand a right from another entity
  - Demand function \( d : TS \to 2^{TxR} \)
  - 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 \( \text{can} \text{create}(a, b) \subseteq TS \times T \)
  - A subject of type \( a \) can create an entity of type \( b \)
- Rule of \text{acyclic creates}
  - Limits the membership in \( \text{can} \text{create}(a, b) \)
  - 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: \( cr(a, b) \subseteq \{ b/r.c \in RL \} \)
  - 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: \( cc(user, file) \) holds
  - Creator can give itself any inert rights: \( cr(user, file) = \{file/r.c\ r \in R\} \)

- **Take-Grant model**
  - A subject can create a subject or an object
    - \( cc(subject, subject) \) and \( cc(subject, object) \) 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)
    - \( cr_s(a, b) = \emptyset; cr_s(a, b) = \{sub/tc, sub/gc, sub/rc, sub/wc\} \)
    - Hence,
      - \( cr(subject, sub) = \{sub/tc, sub/gc, sub/rc, sub/wc\} \emptyset \)
      - \( cr(subject, obj) = \{obj/tc, obj/gc, obj/rc, obj/wc\} \emptyset \)

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