# Decidability

January 16, 2014

1. Security
   - Mono-operational command case
   - General case

2. Protection Systems
   - Take-Grant Systems
   - SPM

# What is "Secure"?

## Leaking

Adding a generic right $r$ where there was not one is *leaking*

## Safe

If a system $S$, beginning in initial state $s_0$, cannot leak right $r$, it is *safe* with respect to the right $r$.

Here, "safe" = "secure" for an abstract model

# What is Does "Decidable" Mean?

## Safety Question

Does there exist an algorithm for determining whether a protection system $S$ with initial state $s_0$ is safe with respect to a generic right $r$?

# Mono-Operational Commands

**Answer:**

Yes!

Proof sketch:
Consider minimal sequence of commands $c_1, \ldots, c_k$ to leak the right

- Can omit **delete**, **destroy**
- Can merge all **create**s into one

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

# Proof (1)

- Consider minimal sequences of commands (of length *m*) needed to leak *r* from system with initial state $s_0$
  - Identify each command by the type of primitive operation it invokes
- Cannot test for *absence* of rights, so **delete**, **destroy** not relevant
  - Ignore them
- Reorder sequences of commands so all **create**s come first
  - Can be done because **enter**s require subject, object to exist
- Commands after these **create**s check only for *existence* of right

# Proof (2)

- It can be shown (see exercise):
  - Suppose $s_1, s_2$ are created, and commands test rights in $A[s_1, o_1]$, $A[s_2, o_2]$
  - Doing the same tests on $A[s_1, o_1]$ and $A[s_1, o_2] = A[s_1, o_2] \cup A[s_2, o_2]$ gives same result
  - Thus all **create**s unnecessary
    - Unless $s_0$ is empty; then you need to create it (1 **create**)
- In $s_0$:
  - $|S_0|$ number of subjects, $|O_0|$ number of objects, $n$ number of (generic) rights
- In worst case, 1 create
  - So a total of at most $(|S_0| + 1)(|O_0| + 1)$ elements
- So $m \leq n(|S_0| + 1)(|O_0| + 1)$

# General Case

**Answer:**

No

Proof sketch:

1. Show arbitrary Turing machine can be reduced to safety problem
2. Then deciding safety problem means deciding the halting problem

# Turing Machine Review

- Infinite tape in one direction
- States $K$, symbols $M$, distinguished blank $b\!\!/$
- State transition function $\delta(k, m) = (k', m', \mathrm{L})$
  in state $k$ with symbol $m$ under the TM head
  replace $m$ with $m'$, move head left one square, enter state $k'$
- Halting state is $q_f$

# Mapping

Turing machine

$$\begin{array}{|c|c|c|c|c}
1 & 2 & 3 & 4 & \cdots \\
\hline
\mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} & \cdots \\
\hline
\multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{\uparrow} & \multicolumn{1}{c}{} & \\
\multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{k} & \multicolumn{1}{c}{} &
\end{array}$$

$\Rightarrow$

access control matrix representation

$$\begin{array}{c|c|c|c|c|c}
 & s_1 & s_2 & s_3 & s_4 & \cdots \\
\hline
s_1 & \mathrm{A} & o & & & \cdots \\
\hline
s_2 & & \mathrm{B} & o & & \cdots \\
\hline
s_3 & & & \mathrm{C}\ k & o & \cdots \\
\hline
s_4 & & & & \mathrm{D}\ e & \cdots \\
\hline
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Turing machine with head over square 3 on tape, in state $k$ and its representation as an access control matrix

$o$ is *own* right

$e$ is *end* right

# Mapping

**Turing machine**

| 1 | 2 | 3 | 4 | $\cdots$ |
|---|---|---|---|---|
| A | B | X | D | $\cdots$ |
| | | | $\uparrow$ $k_1$ | |

$\Rightarrow$

**access control matrix representation**

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $\cdots$ |
|---|---|---|---|---|---|
| $s_1$ | A | $o$ | | | $\cdots$ |
| $s_2$ | | B | $o$ | | $\cdots$ |
| $s_3$ | | | X | $o$ | $\cdots$ |
| $s_4$ | | | | D $k_1$ $e$ | $\cdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ |

After $\delta(k, \text{C}) = (k_1, \text{X}, \text{R})$, where $k$ is the previous state and $k_1$ the current state

# Command Mapping

$\delta(k, \mathrm{C}) = (k_1, \mathrm{X}, \mathrm{R})$ at intermediate becomes:

```
command c_{k,C}(s_i, s_{i+1})
if o in A[s_i, s_{i+1}] and k in A[s_i, s_i] and C in A[s_i, s_i]
then
    delete k from A[s_i, s_i];
    delete C from A[s_i, s_i];
    enter X into A[s_i, s_i];
    enter k_1 into A[s_{i+1}, s_{i+1}];
end
```

# Mapping

Turing machine

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| A | B | X | Y | $\not{b}$ |
| | | | | $\uparrow$ $k_2$ |

$\Rightarrow$

access control matrix representation

| | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ |
|---|---|---|---|---|---|
| $s_1$ | A | $o$ | | | |
| $s_2$ | | B | $o$ | | |
| $s_3$ | | | X | $o$ | |
| $s_4$ | | | | Y | $o$ |
| $s_5$ | | | | | $k_2$ $e$ |

After $\delta(k_1, \text{D}) = (k_2, \text{Y}, \text{R})$, where $k_1$ is the previous state and $k_2$ the current state

# Command Mapping

$\delta(k_1, \mathrm{D}) = (k_2, \mathrm{Y}, \mathrm{R})$ at intermediate becomes:

**command** crightmost$_{k,D}(s_i, s_{i+1})$
**if** $e$ **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** $e$ **from** A$[s_i, s_i]$;
  **create subject** $s_{i+1}$;
  **enter** $o$ **into** A$[s_i, s_{i+1}]$;
  **enter** $e$ **into** A$[s_{i+1}, s_{i+1}]$;
  **delete** $k_1$ **from** A$[s_i, s_i]$;
  **delete** $D$ **from** A$[s_i, s_i]$;
  **enter** Y **into** A$[s_i, s_i]$;
  **enter** $k_2$ **into** A$[s_{i+1}, s_{i+1}]$;
**end**

# Rest of Proof

- Protection system exactly simulates a Turing machine
  - Exactly 1 *end* ($e$) right in access control matrix
  - 1 right in entries corresponds to state
  - Thus, at most 1 applicable command
- If Turing machine enters state $q_f$, then right has leaked
- If safety question decidable, then represent TM as protection system and determine if $q_f$ leaks
  - This implies halting problem is 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; safety question is still undecidable
  - Such systems are called *monotonic*
- 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

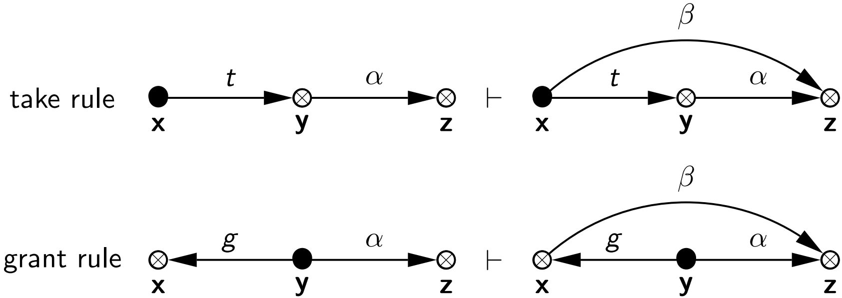
- A specific (not generic) system
  - Set of rules for state transitions
- Safety decidable, and in time linear with the size of the system
- Goal: find conditions under which rights can be transferred from one entity to another in the system

# System

| | |
|---|---|
| ○ | objects (passive entities like files, . . . ) |
| ● | subjects (active entities like users, processes . . . ) |
| $\otimes$ | don't care (either a subject or an object) |
| $G \vdash_x G'$ | apply rewriting rule $x$ (witness) to $G$ to get $G'$ |
| $G \vdash^* G'$ | apply a sequence of rewriting rules (witness) to $G$ to get $G'$ |
| $R = \{t, g, \ldots\}$ | set of rights |

# Take, Grant Rules

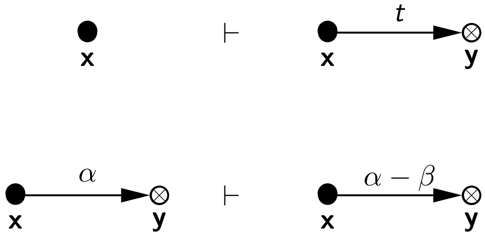
In these rules, $\beta \subseteq \alpha \subseteq R$



# Create, Remove Rules

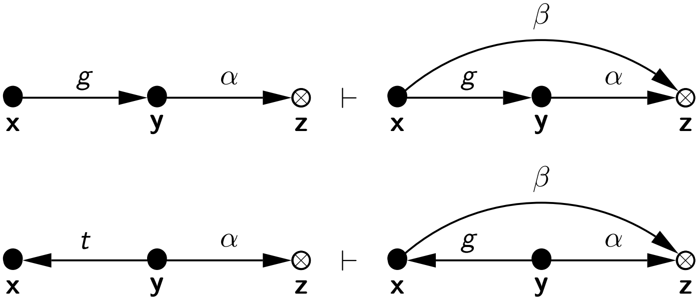
create rule

remove rule

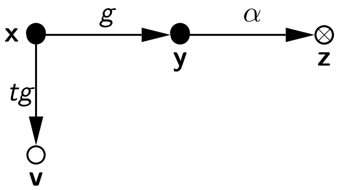



These four rules are the *de jure* rules

# Symmetry of Take and Grant

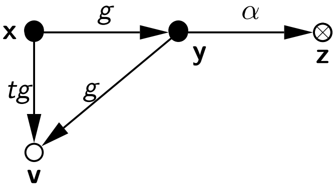


# Symmetry of Take and Grant



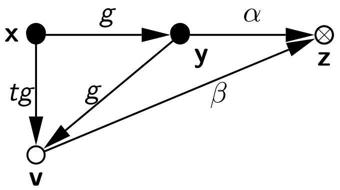

1. **x** creates ($tg$ to new) **v**

# Symmetry of Take and Grant



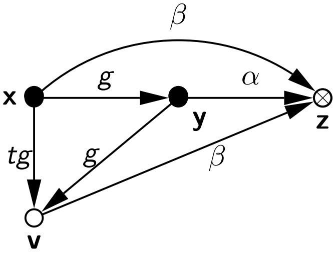

1. **x** creates ($tg$ to new) **v**
2. **x** grants ($g$ to **v**) to **y**

# Symmetry of Take and Grant




1. **x** creates ($tg$ to new) **v**
2. **x** grants ($g$ to **v**) to **y**
3. **y** grants ($\beta$ to **z**) to **v**

# Symmetry of Take and Grant




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

# Islands

- *tg-path*: path of distinct vertices connected by edges labeled *t* or *g*
  - Call them *tg-connected*
- *island*: maximal *tg*-connected subject-only subgraph
  - Any right that a vertex in the island has, can be shared with any other vertex in the island

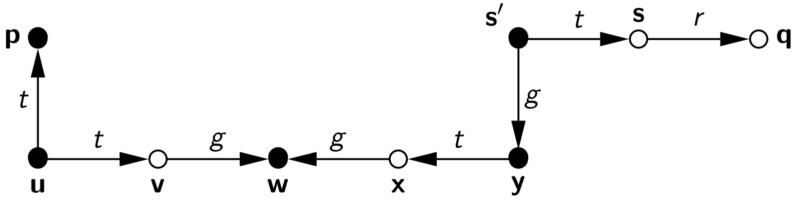
# Initial, Terminal Spans

- *initial span* from **x** to **y**: **x** can give rights it has to **y**
  - **x**subject
  - *tg*-path between **x**, **y** with word in $\{\overrightarrow{t^*}\overrightarrow{g}\} \cup \{\nu\}$
- *terminal span* from **x** to **y**: **x** can get rights **y** has
  - **x**subject
  - *tg*-path between **x**, **y** with word in $\{\overrightarrow{t^*}\} \cup \{\nu\}$

# Bridges

- *bridge* *tg*-path between subjects **x**, **y**, with associated word in $\{\overrightarrow{t^*}, \overleftarrow{t^*}, \overrightarrow{t^*}\overrightarrow{g}\overleftarrow{t^*}, \overrightarrow{t^*}\overleftarrow{g}\overleftarrow{t^*}\}$
  - rights can be transferred between the two endpoints
  - *not* an island as intermediate vertices are objects

# Example




- islands: $\{\mathbf{p}, \mathbf{u}\}, \{\mathbf{w}\}, \{\mathbf{y}, \mathbf{s}'\}$
- bridges: **u**, **v**, **w**; **w**, **x**, **y**
- initial span: **p** (associated word $\nu$)
- terminal span: $\mathbf{s}'\mathbf{s}$ (associated word $\overrightarrow{t}$)

# *can·share* Predicate

*can·share*($r$, **x**, **y**, $G_0$) holds 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 *de jure* rules and in $G_n$ there is an edge from **x** to **y** labeled $r$

# *can·share* Theorem

*can·share*($r$, $\mathbf{x}$, $\mathbf{y}$, $G_0$) holds if, and only if, there is an edge from $\mathbf{x}$ to $\mathbf{y}$ labeled $r$ in $G_0$, or the following hold simultaneously:

- there is an $\mathbf{s}$ in $G_0$ with an $\mathbf{s}$-to-$\mathbf{y}$ edge labeled $r$;
- there is a subject $\mathbf{x}' = \mathbf{x}$ or $\mathbf{x}'$ initially spans to $\mathbf{x}$;
- there is a subject $\mathbf{s}' = \mathbf{s}$ or $\mathbf{s}'$ terminally spans to $\mathbf{s}$; and
- there are islands $I_1, \ldots, I_k$ connected by bridges, $\mathbf{x}'$ is in $I_1$, and $\mathbf{s}'$ is in $I_k$

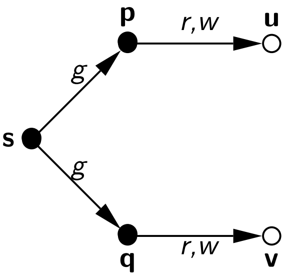
# Outline of Proof

1. **s** has *r* rights over **y**
2. **s**′ acquires *r* rights over **y** from **s**
   - Definition of terminal span
3. **x**′ acquires *r* rights over **y** from **s**′
   - Repeated application of sharing among vertices in islands, passing rights along bridges
4. **x**′ gives *r* rights over **y** to **x**
   - Definition of initial span

# Interpretation

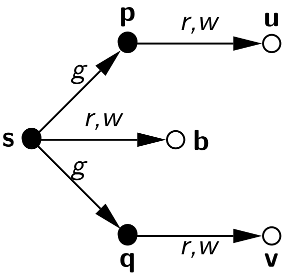
- Access control matrix is generic
  - Can be applied in any situation
- Take-Grant has specific rules, rights
  - Can be applied in situations matching rules, rights
- What states can evolve from a system that is modeled using the Take-Grant Protection Model?

# Example: Shared Buffer




Goal: **p**, **q** to communicate through shared buffer **b** controlled by trusted entity **s**

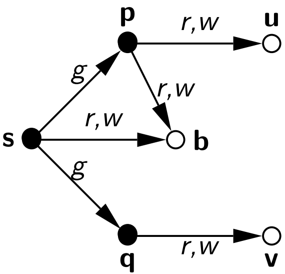
# Example: Shared Buffer




Goal: **p**, **q** to communicate through shared buffer **b** controlled by trusted entity **s**

1. **s** creates ($\{r, w\}$ to) new object **b**

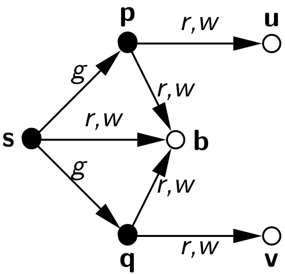
# Example: Shared Buffer




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**

# 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**

# Schematic Protection 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 $\mathbf{X}/r$, where $\mathbf{X}$ is entity and $r$ right
- Functions determine rights transfer
  - Link: are source, target "connected"?
  - Filter: is transfer of ticket authorized?

# Link Predicate

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

# Schematic Protection Model

- Take-Grant:
  $$link(\mathbf{X}, \mathbf{Y}) = \mathbf{Y}/g \in dom(\mathbf{X}) \vee \mathbf{X}/t \in dom(\mathbf{Y})$$
- Broadcast:
  $$link(\mathbf{X}, \mathbf{Y}) = \mathbf{X}/b \in dom(\mathbf{X})$$
- Pull:
  $$link(\mathbf{X}, \mathbf{Y}) = \mathbf{Y}/p \in dom(\mathbf{Y})$$

# Filter Function

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

# Examples

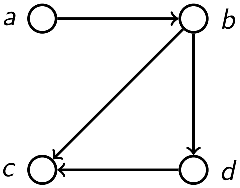
- $f_i(\tau(\mathbf{Y}), \tau(\mathbf{Z})) = T \times R$
  Any ticket can be transferred (if other conditions met)
- $f_i(\tau(\mathbf{Y}), \tau(\mathbf{Z})) = T \times RI$
  Only tickets with inert rights can be transferred (if other conditions met)
- $f_i(\tau(\mathbf{Y}), \tau(\mathbf{Z})) = \varnothing$
  No tickets can be transferred

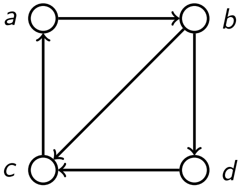
# Example: Take-Grant Model

- $TS$ = { subjects }, $TO$ = { objects }
- $RC$ = { $tc$, $gc$ }, $RI$ = { $rc$, $wc$, ...}
- $link(\mathbf{p}, \mathbf{q}) = \mathbf{p}/t \in dom(\mathbf{q}) \vee \mathbf{q}/g \in dom(\mathbf{p})$
- $f$(subject, subject) = { subject, object } $\times$ { $tc$, $gc$, $rc$, $wc$ }

# Create Operation

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



# 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 $\mathbf{B}/r{:}c$ if and only if $b/r{:}c \in cr(a, b)$
- **B**subject: $cr(a, b)$ has 2 subsets
  - $cr_P(a, b)$ added to **A**, $cr_C(a, b)$ added to **B**
  - **A** gets $\mathbf{B}/r{:}c$ if and only if $b/r{:}c \in cr_P(a, b)$
  - **B** gets $\mathbf{A}/r{:}c$ if and only if $a/r{:}c \in cr_C(a, b)$

# Non-Distinct Types

- $cr(a, a)$: who gets what?
  - $\mathit{self}/r{:}c$ are tickets for creator
  - $a/r{:}c$ are tickets for created entity

$$cr(a, a) = \{\ a/r{:}c, \mathit{self}/r{:}c \ \mid\ r{:}c \in R\ \}$$

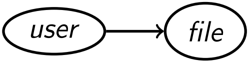
# Attenuating Create Rule

$cr(a, b)$ is attenuating if:

1. $cr_C(a, b) \subseteq cr_P(a, b)$ and
2. $a/r{:}c \in cr_P(a, b) \Rightarrow self/r{:}c \in cr_P(a, b)$

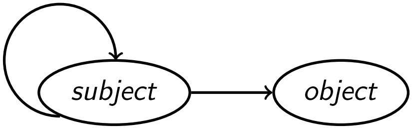
# Example: Owner-Based Policy

- Users can create files, creator can give itself any inert rights over file
  - $cc = \{(user, file)\}$
  - $cr(user, file) = \{\ file/r{:}c \quad | \quad r \in RI\ \}$
- 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
  - $cc = \{(s, s), (s, o)\}$
  - $cr_C(a, b) = \varnothing$
  - $cr_P(s, s) = \{s/tc, s/gc, s/rc, s/wc\}$
  - $cr_P(s, o) = \{o/rc, o/wc\}$
- 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 in 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 $path^h(\mathbf{X}, \mathbf{Y})$

- $\mathbf{X}$, $\mathbf{Y}$ connected by one link or a sequence of links
- Formally, either of these hold:
  - For some $i$, $link_i^h(\mathbf{X}, \mathbf{Y})$; or
  - There is a sequence of subjects $\mathbf{X}_0, \ldots, \mathbf{X}_n$ such that $link_i^h(\mathbf{X}, \mathbf{X}_0)$, $link_i^h(\mathbf{X}_n, \mathbf{Y})$, and for $k = 1, \ldots, n$, $link_i^h(\mathbf{X}_{k-1}, \mathbf{X}_k)$
- If multiple such paths, refer to $path_j^h(\mathbf{X}, \mathbf{Y})$

# Capacity $cap(path^h(\mathbf{X}, \mathbf{Y}))$

- Set of tickets that can flow over $path^h(\mathbf{X}, \mathbf{Y})$
  - If $link_i^h(\mathbf{X}, \mathbf{Y})$: set of tickets that can be copied over the link (i.e., $f_i(\tau(\mathbf{X}), \tau(\mathbf{Y}))$)
  - Otherwise, set of tickets that can be copied over all links in the sequence of links making up the $path^h(\mathbf{X}, \mathbf{Y})$
- 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 $\mathbf{X}$ and $\mathbf{Y}$ in state $h$. Then *flow function*

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

is:

$$flow^h(\mathbf{X}, \mathbf{Y}) = \bigcup_{i=1,\ldots,m} cap(path_i^h(\mathbf{X}, \mathbf{Y}))$$

# Properties of Maximal State

- Maximizes flow between all pairs of subjects
  - State is called *
  - Ticket in $flow^*(\mathbf{X}, \mathbf{Y})$ 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 of Maximal State

- Definition: $g \leq_0 h$ holds iff for all $\mathbf{X}, \mathbf{Y} \in SUB^0$, $flow^g(\mathbf{X}, \mathbf{Y}) \subseteq flow^h(\mathbf{X}, \mathbf{Y})$
  - Note: if $g \leq_0 h$ and $h \leq_0 g$, then $g$, $h$ are equivalent states
  - 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, answering first question (uniqueness of maximal state)

# Useful Lemma

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

# Proof of Useful Lemma

By induction on the size of $H$

BASIS: For $H = \varnothing$, $|H| = 0$, claim is trivially true

INDUCTION HYPOTHESIS: For $|H| = n$, claim holds

INDUCTION STEP: $|H'| = n + 1$, where $H' = G \cup \{h\}$. By hypothesis, there is a $g \in G$ such that $x \leq_0 g$ for all $x \in G$
Let $M$ be an interleaving of histories of $g$, $h$, which:

- Preserves relative order of transitions in $g$, $h$
- Omits second create operation if duplicated

$M$ ends up in state $m$
If $path^g(\mathbf{X}, eY)$ for $\mathbf{X}, \mathbf{Y} \in SUB^g$, $path^m(\mathbf{X}, \mathbf{Y})$, so $g \leq_0 m$
If $path^h(\mathbf{X}, eY)$ for $\mathbf{X}, \mathbf{Y} \in SUB^h$, $path^m(\mathbf{X}, \mathbf{Y})$, so $h \leq_0 m$
Hence $m$ is a maximal state in $H'$

# Answer to “Does Every System Have a Maximal State”

Theorem: every system has a maximal state *

Outline of proof: Let $K$ be the set of derivable states containing exactly one state from each equivalence class of derivable states

- Let $\mathbf{X}, \mathbf{Y} \in SUB^0$.
- Flow function's range is $2^{T \times R}$, so it can take on at most $|2^{T \times R}|$ values.
- There are $|SUB^0|^2$ pairs of subjects in $SUB^0$
- So at most $|2^{T \times R}| \; |SUB^0|^2$ distinct equivalence classes
- So $K$ is finite

So the lemma's conditions hold, giving the answer “yes”