Lecture 3: Decidability

January 11, 2011

1 Review

2 Decidability of security
- Mono-operational command case
- General case

3 Take-Grant Protection Model
- Sharing rights
- Take-Grant Systems
- Stealing rights
- Conspiracy

Why no "or"?

- Unnecessary!
- Break conditional expression into sequence of disjuncts
- Write command with same body for each disjunct
- Call them sequentially!

r, c Commands

```
command grant·read·file·ifr(p, f)
    if r in A[p, f]
    then
        enter r into A[q, f];
        enter w into A[q, f];
end
command grant·read·file·ifc(p, f)
    if c in A[p, f]
    then
        enter r into A[q, f];
        enter w into A[q, f];
end
```

r or *c* Command

```
command $grant \cdot read \cdot file \cdot ifrorc(p, f)$
    $grant \cdot read \cdot file \cdot ifr(p, f)$
    $grant \cdot read \cdot file \cdot ifc(p, f)$
end
```

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 homework):
  - 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

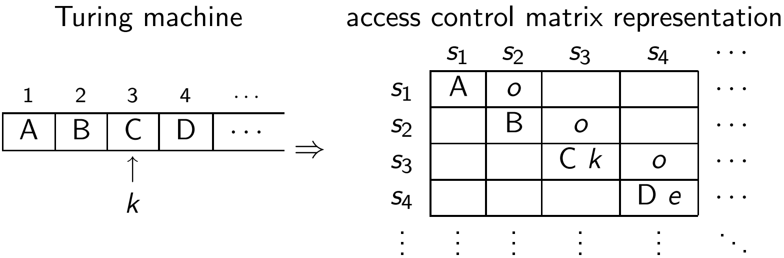
Proof sketch:

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 $\not b$
- 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

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

access control matrix representation

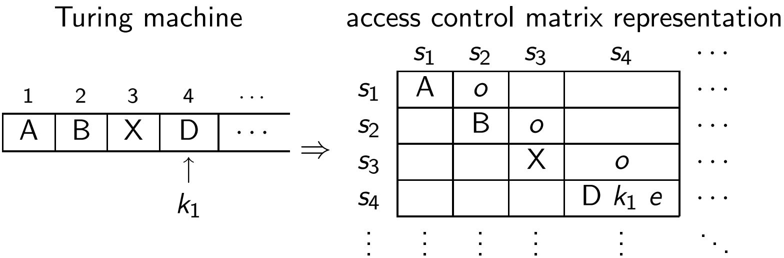
$$\begin{array}{c|c|c|c|c|c} & s_1 & s_2 & s_3 & s_4 & \cdots \\ \hline s_1 & A & o & & & \cdots \\ \hline s_2 & & B & o & & \cdots \\ \hline s_3 & & & C\ k & o & \cdots \\ \hline s_4 & & & & D\ e & \cdots \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

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

$$\begin{array}{c|c|c|c|c|c}
 & s_1 & s_2 & s_3 & s_4 & \cdots \\
\hline
s_1 & \mathrm{A} & o & & & \cdots \\
\hline
s_2 & & \mathrm{B} & o & & \cdots \\
\hline
s_3 & & & \mathrm{X} & o & \cdots \\
\hline
s_4 & & & & \mathrm{D}\ k_1\ e & \cdots \\
\hline
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

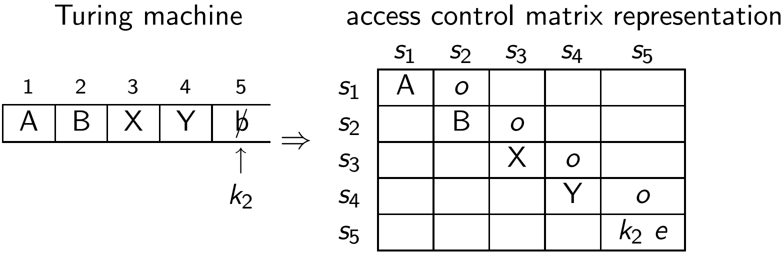
After $\delta(k, \mathrm{C}) = (k_1, \mathrm{X}, \mathrm{R})$, where k is the previous state and k_1 the current state

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	$\not b$

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 $y$;
    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 symbols is recursively enumerable
- Delete **create** primitive; then safety question is complete in **P-SPACE**
- Delete **destroy**, **delete** primitives; then safety question is 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

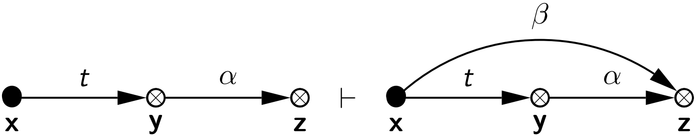
System

○	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

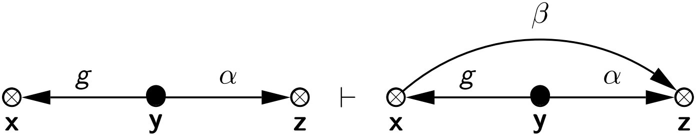
Take, Grant Rules

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

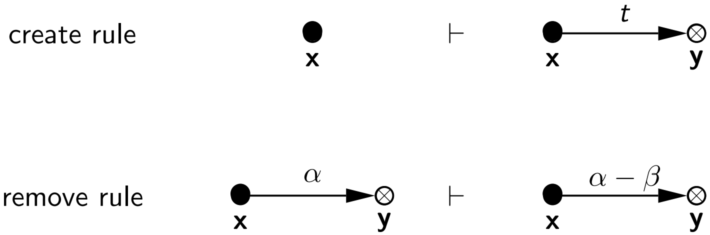
take rule



grant rule

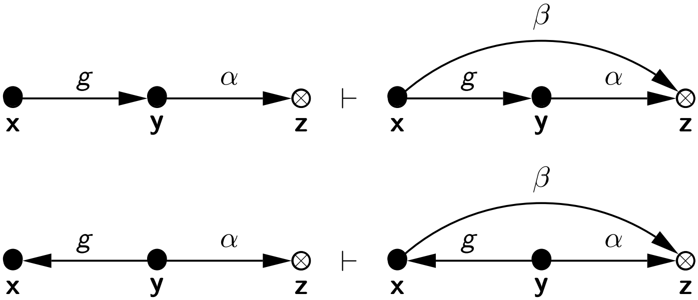


Create, Remove Rules

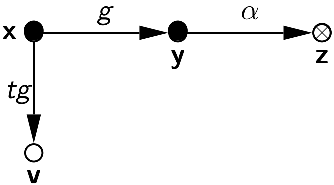


These four rules are the *de jure* rules

Symmetry of Take and Grant

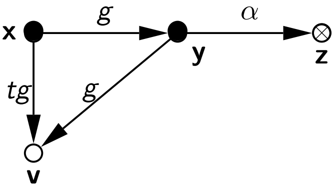


Symmetry of Take and Grant



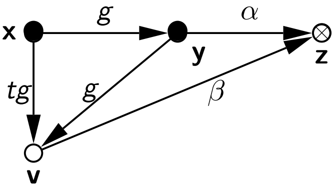
1. **x** creates (tg to new) **v**

Symmetry of Take and Grant



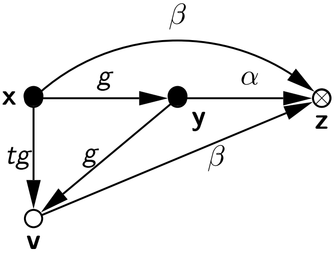
1. **x** creates (tg to new) **v**
2. **x** grants (g to **v**) to **y**

Symmetry of Take and Grant



1. **x** creates (tg to new) **v**
2. **x** grants (g to **v**) to **y**
3. **y** grants (β 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 (β to **z**) to **v**
4. **x** takes (β 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

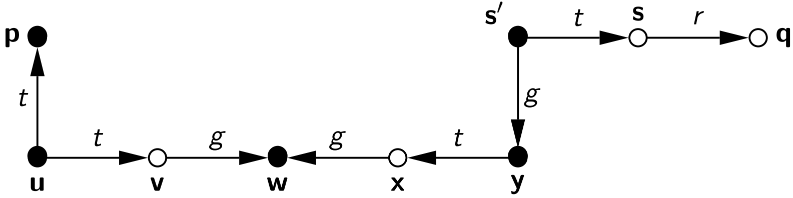
Initial, Terminal Spans

- *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 ν)
- 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, **x**, **y**, G_0) holds 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 $\mathbf{x}' = \mathbf{x}$ or $\mathbf{x}'$ initially spans to **x**;
- there is a subject $\mathbf{s}' = \mathbf{s}$ or $\mathbf{s}'$ terminally spans to **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. $\mathbf{s}$ has r rights over $\mathbf{y}$
2. $\mathbf{s}'$ acquires r rights over $\mathbf{y}$ from $\mathbf{s}$
   - Definition of terminal span
3. $\mathbf{x}'$ acquires r rights over $\mathbf{y}$ from $\mathbf{s}'$
   - Repeated application of sharing among vertices in islands, passing rights along bridges
4. $\mathbf{x}'$ gives r rights over $\mathbf{y}$ to $\mathbf{x}$
   - 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?

Take-Grant Generated Systems

Theorem: Let G_0 be a protection graph with 1 subject and no edges. Let R be a set of rights. Then $G_0 \vdash^* G$ if, and only if,

- G is a finite, directed graph consisting of subjects, objects, and edges;
- the edges are labeled from a non-empty subset of R; and
- at least 1 vertex in G has no incoming edges

Proof (1)

⇒: By construction; let G be the final graph in the theorem

- Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be subjects in G
- Let $\mathbf{x}_1$ have no incoming edges
- Let $\alpha = R$

Construct G' as follows:

1. Do "$\mathbf{x}_1$ creates ($\alpha \cup \{g\}$ to) new subject $\mathbf{x}_i$"
2. For all $(\mathbf{x}_i, \mathbf{x}_j)$ where $\mathbf{x}_i$ has a right over $\mathbf{x}_j$, do "$\mathbf{x}_1$ grants (α to $\mathbf{x}_j$) to $\mathbf{x}_i$"
3. Let β be the rights $\mathbf{x}_i$ has over $\mathbf{x}_j$ in G; then do "$\mathbf{x}_1$ removes (($\alpha \cup \{g\}) - \beta$) to $\mathbf{x}_j$)"

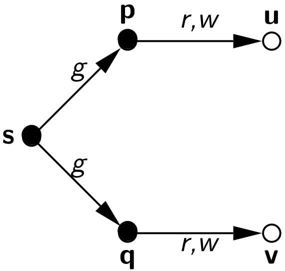
Now G' is the desired G

Proof (2)

$\Leftarrow$: Let **v** be the initial subject, and $G_0 \vdash^* G$

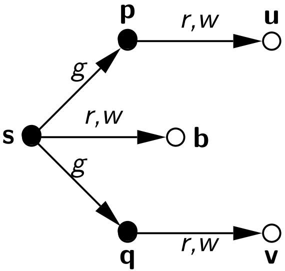
- Inspection of rules gives:
  - G is finite;
  - G is a directed graph;
  - Subjects and objects only; and
  - All edges are labeled with nonempty subsets of R
- Limits of rules:
  - None allows vertices to be deleted, so **v** is in G
  - None adds *incoming* edges to vertices without any incoming edges, so **v** has no incoming edges.

Example: Shared Buffer



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

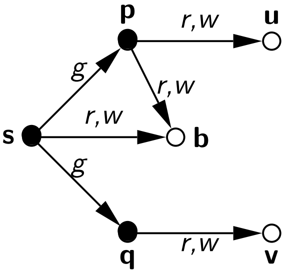
Example: Shared Buffer



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

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

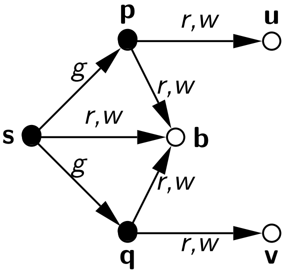
Example: Shared Buffer



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**

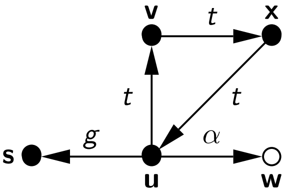
1. **s** creates ($\{r, w\}$ to) new object **b**
2. **s** grants ($\{r, w\}$ to **b**) to **p**
3. **s**grants ($\{r, w\}$ to **b**) to **q**

can·steal Predicate

can·steal(r, **x**, **y**, G_0) holds if, and only if, there is no edge from **x** to **y** labeled r in G_0, and the following hold simultaneously:

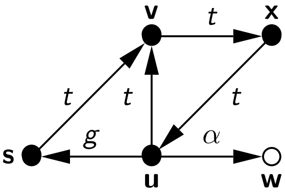
- there is an edge from **x** to **y** labeled r in G;
- there is a sequence of rule applications $\rho_1, \ldots, \rho_n$ such that $G_{i-1} \vdash_{\rho_i} G_i$; and
- for all vertices **v**, **w** in G_{i-1}, if there is an edge from **v** to **y** in G_0 labeled r, then ρ_i is *not* of the form "**v** grants (r to **y**) to **w**"

Example of Stealing



can·steal$(\alpha, \mathbf{s}, \mathbf{w}, G_0)$

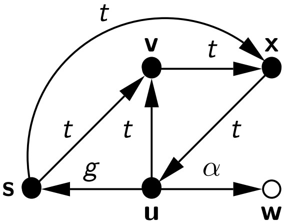
Example of Stealing



can·steal(α, **s**, **w**, G_0):

1. **u** grants (t to **v**) to **s**

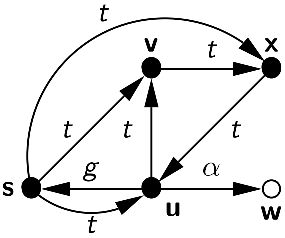
Example of Stealing



can·steal(α, **s**, **w**, G_0):

1. **u** grants (t to **v**) to **s**
2. **s** takes (t to **x**) from **v**

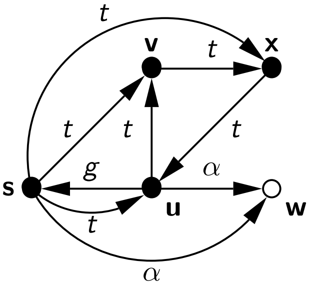
Example of Stealing



can·steal(α, **s**, **w**, G_0):

1. **u** grants (t to **v**) to **s**
2. **s** takes (t to **x**) from **v**
3. **s** takes (t to **u**) from **x**

Example of Stealing



can·steal(α, **s**, **w**, G_0):

1. **u** grants (t to **v**) to **s**
2. **s** takes (t to **x**) from **v**
3. **s** takes (t to **u**) from **x**
4. **s** takes (α to **w**) from **u**

can·steal Theorem

can·steal(α, **x**, **y**, G_0) holds if, and only if, the following hold simultaneously:

- there is no edge from **x**-to-**y** labeled α in G_0;
- there is a subject $\mathbf{x}' = \mathbf{x}$ or $\mathbf{x}'$ initially spans to **x**;
- there is a vertex **s** with an edge to **y** labeled α in G_0; and
- *can·share*(t, $\mathbf{x}'$, **s**, G_0) holds

Proof (1)

$\Rightarrow$: Assume all four conditions hold

- If **x** a subject:
  - **x** gets t rights to **s** (last condition); then takes α to **y** from **s** (third condition)
- If **x** an object:
  - *can·share*(t, $\mathbf{x}'$, **s**, G_0) holds
  - If $\mathbf{x}'$ has no α edge to **y** in G_0, $\mathbf{x}'$ takes (α to **y**) from **s** and grants it to **x**
  - If $\mathbf{x}'$ has an edge to **y** in G_0, $\mathbf{x}'$ creates surrogate $\mathbf{x}''$, gives it (t to **s**) and (g to $\mathbf{x}''$); then $\mathbf{x}''$ takes (α to **y**) and grants it to **x**

Proof (2)

$\Leftarrow$: Assume *can·steal*(α, **x**, **y**, G_0) holds

- First two conditions are immediate from definition of *can·share*, *can·steal*
- Third condition is immediate from theorem of conditions for *can·share*
- Fourth condition: let ρ be a minimal length sequence of rule applications deriving G_n from G_0
    - Let i be the smallest index such that $G_{i-1} \vdash_{\rho_i} G_i$ that adds α from some **p** to **y** in G_i
    - What rule is ρ_i?

Proof (3)

- Not remove or create rule
  - **y** exists already
- Not grant rule
  - G_i is the first graph in which an edge labeled α to **y** is added, so by definition of *can·share*, it cannot be a grant
- Therefore ρ_i must be a take rule, so *can·share*(t, **p**, **s**, G_0) holds
  - By earlier theorem, there is a subject $\mathbf{s}'$ such that $\mathbf{s}' = \mathbf{s}$ or $\mathbf{s}'$ terminally spans to **s**
  - Also, sequence of islands $I_1, \ldots, I_n$ with $\mathbf{x}' \in I_1$, $\mathbf{s}' \in I_n$
- Now consider what **s** is

Proof (4)

- If $\mathbf{s}$ object, $\mathbf{s}' \neq \mathbf{s}$
  - If $\mathbf{s}'$, $\mathbf{p}$ in same island, take $\mathbf{p} = \mathbf{s}'$; the $\textit{can}\cdot\textit{share}(t, \mathbf{x}, \mathbf{s}, G_0)$ holds
  - If they are not, the sequence is minimal, contradicting assumption
  - So choose $\mathbf{s}'$ in same island as $\mathbf{p}$

Proof (5)

If **s** subject, $\mathbf{p} \in I_n$

- If $\mathbf{p} \notin G_0$, there is a subject **q** such that *can·share*(t, **q**, **s**, G_0) holds
    - $\mathbf{s} \in G_0$ and none of the rules add new lables to incoming edges on existing vertices
- As **s** owns α rights to **y** in G_0, two cases arise:
    - If $\mathbf{s} = \mathbf{q}$, replace "**s** grants (α to **y**) to **q**" with the sequence:
      p takes (α to **y**) from **s**
      p takes (g to **q**) from **s**
      p grants (α to **y**) to **q**
    - If $\mathbf{s} = \mathbf{q}$, you only need the first

Conspiracy

- Minimize number of actors to generate a witness for *can·share*(α, **x**, **y**, G_0)
  - *Actor* is defined as **x** such that **x** initiates ρ_i
- Access set describes the “reach” of a subject
- Deletion set is set of verticies that cannot be involved in a transfer of rights
- Build *conspiracy graph* to capture how rights flow, and derive actors from it

Access Set

- *Access set* $A(\mathbf{x})$ *with focus* **x**: set of vertices
    - $\{\mathbf{x}\}$
    - $\{\mathbf{y} \mid \mathbf{x}$ initially spans to $\mathbf{y}\}$
    - $\{\mathbf{y} \mid \mathbf{x}$ terminally spans to $\mathbf{y}\}$
- Idea is that vertex at focus can give rights to, or acquire rights from, a vertex in access set