Introduction to Computer Security

Lecture 2

September 4, 2003

Protection System

- Subject (S: set of all subjects)
  - Active entities that carry out an action/operation on other entities; Eg.: users, processes, agents, etc.
- Object (O: set of all objects)
  - Eg.:Processes, files, devices
- Right
  - An action/operation that a subject is allowed/disallowed on objects

Access Control Matrix Model

- Access control matrix
  - Describes the protection state of a system.
  - Characterizes the rights of each subject
  - Elements indicate the access rights that subjects have on objects
- ACM is an abstract model
  - Rights may vary depending on the object involved
- ACM is implemented primarily in two ways
  - Capabilities (rows)
  - Access control lists (columns)

State Transitions

- Let initial state $X_0 = (S_0, O_0, A_0)$
- Notation
  - $X_i + \tau_{i+1}\ X_{i+1}$: upon transition τ_{i+1}, the system moves from state X_i to X_{i+1}
  - $X +^{*}\ Y$: the system moves from state X to Y after a set of transitions
  - $X_i +\ c_{i+1}\ (p_{i+1,1}, p_{i+1,2}, \ldots, p_{i+1,\mathrm{m}})\ X_{i+1}$: state transition upon a command
- For every command there is a sequence of state transition operations

Primitive commands (HRU)

Create subject s	Creates new row, column in ACM;
Create object o	Creates new column in ACM
Enter r into $a[s, o]$	Adds *r* right for subject *s* over object *o*
Delete r from $a[s, o]$	Removes *r* right from subject *s* over object *o*
Destroy subject s	Deletes row, column from ACM;
Destroy object o	Deletes column from ACM

System commands using primitive operations

- process p creates file f with owner *read* and *write* (r, w) will be represented by the following:

```
Command $create\_file(p, f)$
  Create object $f$
  Enter $own$ into $a[p,f]$
  Enter $r$ into $a[p,f]$
  Enter $w$ into $a[p,f]$
End
```

- Defined commands can be used to update ACM

```
Command $make\_owner(p, f)$
  Enter $own$ into $a[p,f]$
End
```

- Mono-operational: the command invokes only one primitive

Conditional Commands

- Mono-operational + mono-conditional

```
Command grant_read_file(p, f, q)
  If own in a[p,f]
  Then
     Enter r into a[q,f]
End
```

- Mono-operational + biconditional

```
Command grant_read_file(p, f, q)
  If r in a[p,f] and c in a[p,f]
  Then
     Enter r into a[q,f]
End
```

- Why not “OR”??

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?

What is a secure system?

- A simple definition
  - A secure system doesn't allow violations of a security policy
- Alternative view: based on distribution of rights to the subjects
  - Leakage of rights: (unsafe with respect to a right)
    - Assume that *A* represents a secure state and a right r is not in any element of A.
    - Right r is said to be leaked, if a sequence of operations/commands adds *r* to an element of *A,* which not containing *r*
- Safety of a system with initial protection state X_o
  - Safe with respect to r: System is *safe with respect to r* if *r* can never be leaked
  - Else it is called unsafe with respect to right r.

Safety Problem: *formally*

- Given
  - initial state $X_0 = (S_0, O_0, A_0)$
  - Set of primitive commands c
  - r is not in $A_0[s, o]$
- Can we reach a state X_n where
  - $\exists s,o$ such that $A_n[s,o]$ includes a right r not in $A_0[s,o]$?

- If so, the system is not safe
- But is “safe” secure?

Decidability Results *(Harrison, Ruzzo, Ullman)*

- Theorem: Given a system where each command consists of a single *primitive* command (mono-operational), there exists an algorithm that will determine if a protection system with initial state X_0 is safe with respect to right *r*.
- Proof: determine minimum commands *k* to leak
  - Delete/destroy: Can’t leak (or be detected)
  - Create/enter: new subjects/objects “equal”, so treat all new subjects as one
  - If *n* rights, leak possible, must be able to leak $n(|S_0|+1)(|O_0|+1)+1$ commands
- Enumerate all possible states to decide

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

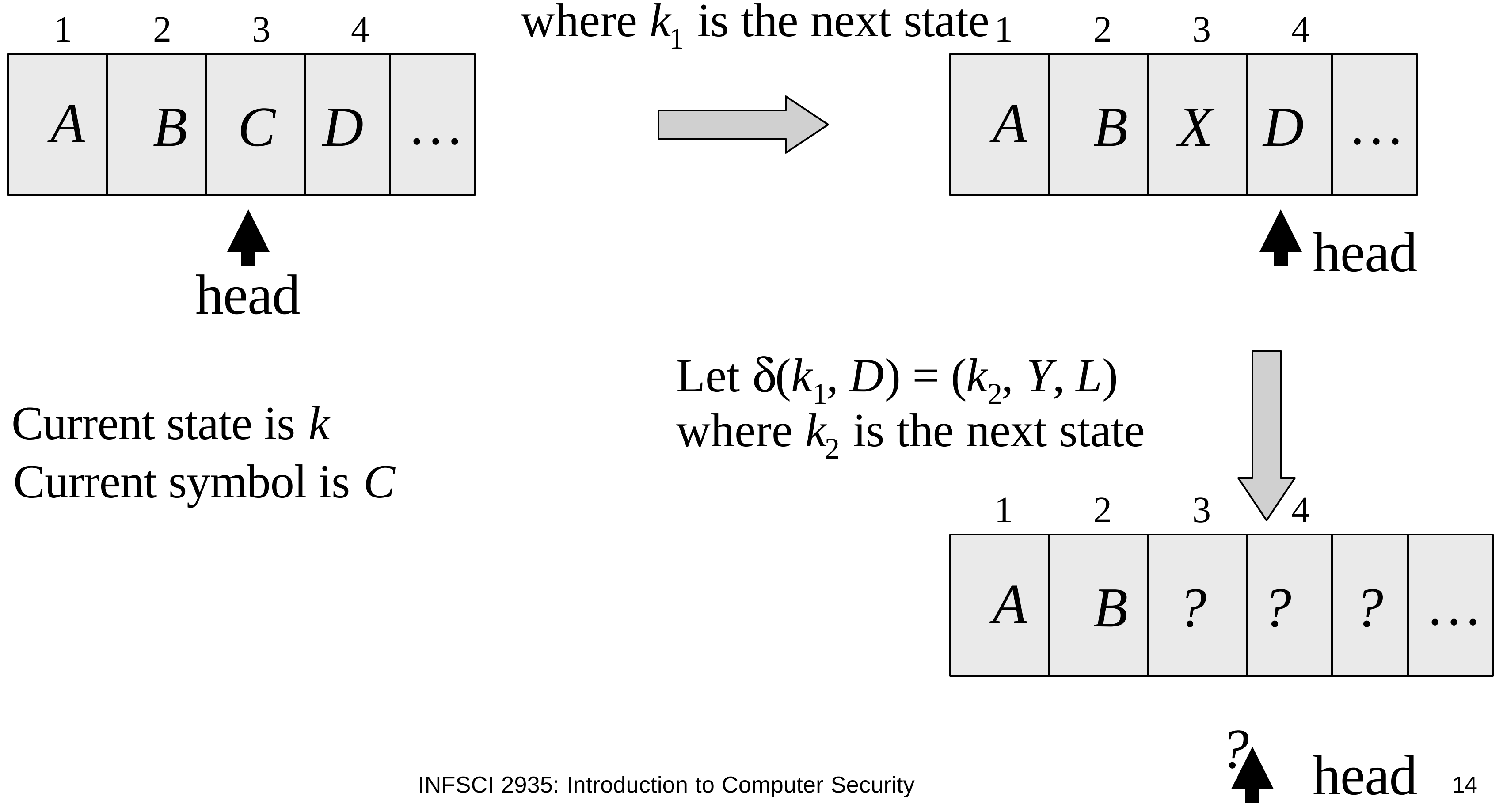
Turing Machine

- 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

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

Let $\delta(k_1, D) = (k_2, Y, L)$
where k_2 is the next state



Current state is k
Current symbol is C

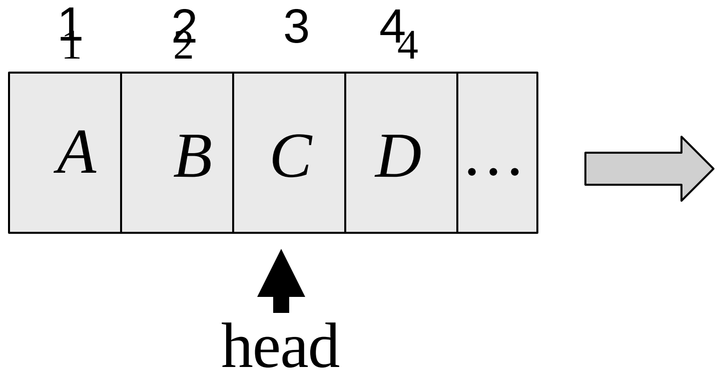
Turing Machine & halting problem

- 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)

General Safety Proble

- 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 $1 = i < k$

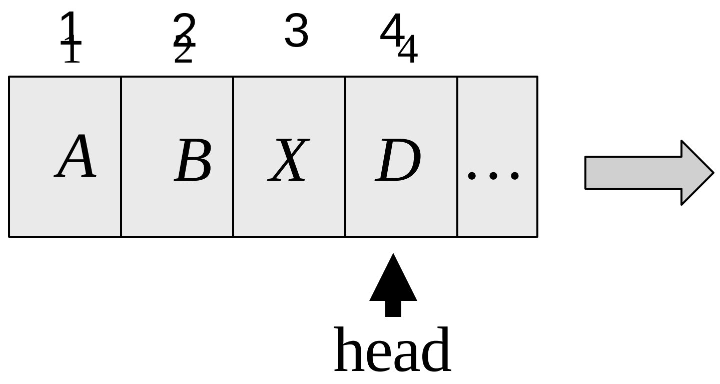
Mapping



	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			C k	*own*	
s_4				D end	

Current state is k
Current symbol is C

Mapping



	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			X	*own*	
s_4				D k_1 end	

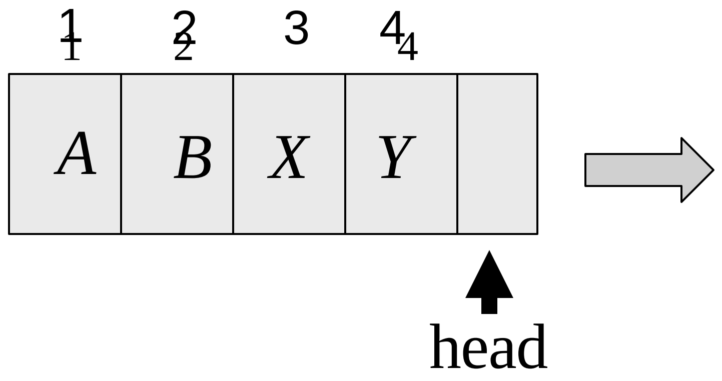
After $\delta(k, \mathrm{C}) = (k_1, \mathrm{X}, \mathrm{R})$ where k is the current state and k_1 the next state

Command Mapping

$\delta(k, \mathrm{C}) = (k_1, \mathrm{X}, \mathrm{R})$

```
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



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

	s_1	s_2	s_3	s_4	s_5
s_1	A	*own*			
s_2		B	*own*		
s_3			X	*own*	
s_4				Y	*own*
s_5					b k_2 end

Command Mapping

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

```
command crightmost$_{k,\mathrm{C}}$($s_4$,$s_5$)
if end in $A[s_4,s_4]$ and $k_1$ in $A[s_4,s_4]$ and D in $A[s_4,s_4]$
then
    delete end from $A[s_4,s_4]$;
    create subject $s_5$;
    enter own into $A[s_4,s_5]$;
    enter end into $A[s_5,s_5]$;
    delete $k_1$ from $A[s_4,s_4]$;
    delete D from $A[s_4,s_4]$;
    enter Y into $A[s_4,s_4]$;
    enter $k_2$ into $A[s_5,s_5]$;
end
```

Rest of Proof

- Similar commands move right, move right at end of tape
  - Refer to book
- 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 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**
  - 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

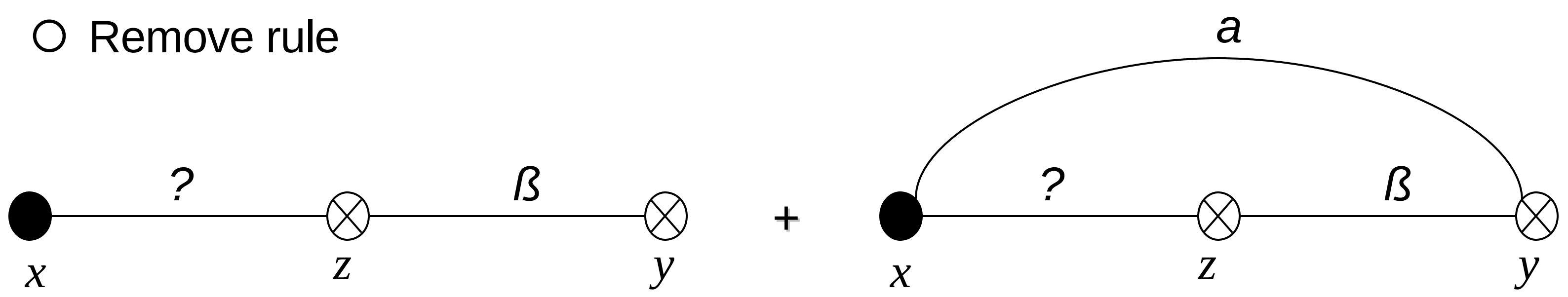
- The safety question for biconditional monotonic protection systems is undecidable
- The safety question for monoconditional, monotonic protection systems is decidable
- The safety question for monoconditional protection systems with **create**, **enter**, **delete** (and no **destroy**) is decidable.
- Observations
  - Safety is undecidable for the generic case
  - Safety becomes decidable when restrictions are applied

What is the implication?

- Safety decidable for some models
  - Are they practical?
- Safety only works if maximum rights known in advance
  - Policy must specify all rights someone could get, not just what they have
  - Where might this make sense?
- Two key questions
  - Given a particular system with specific rules for transformation, can we show that the safety question is decidable?
    - E.g. Take-grant model
  - What are the weakest restrictions that will make the safety question decidable in that system

Take-Grant Protection Model

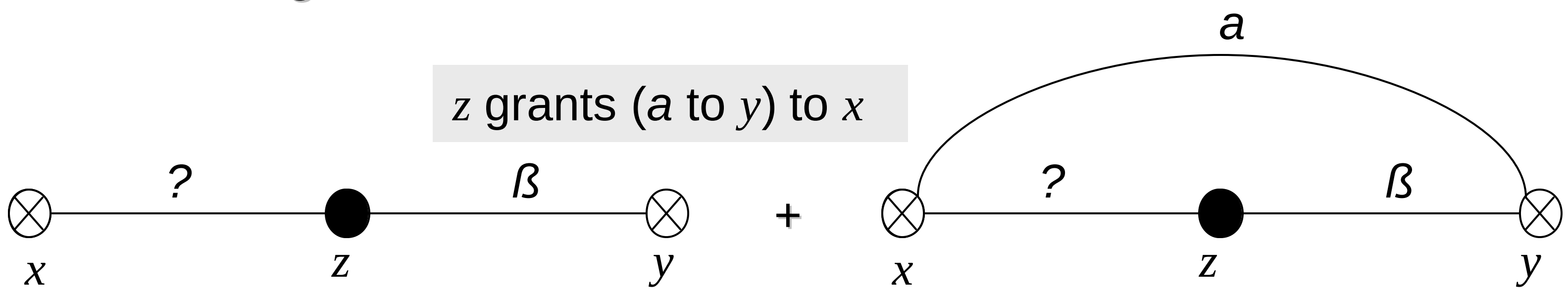
- System is represented as a directed graph
  - Subject: ●
  - Object: ○
  - Either: ⊗
  - Labeled edge indicate the rights that the source object has on the destination object
- Four graph rewriting rules ("de jure", "by law", "by rights")
  - Take rule
  - Grant rule
  - Create rule
  - Remove rule



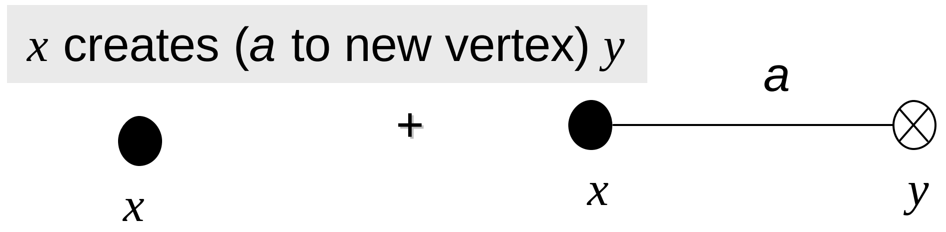
x takes (*a* to *y*) from *z*

Take-Grant Protection Model

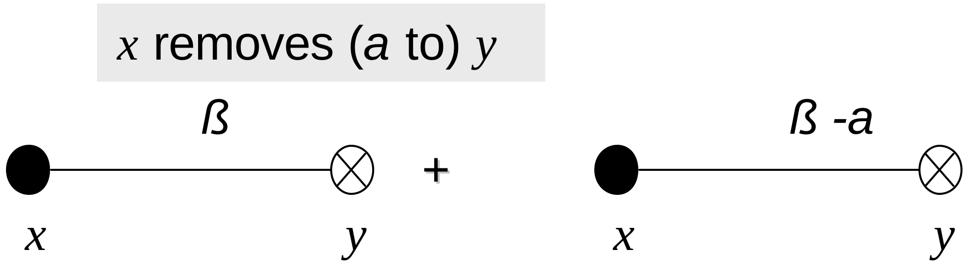
2. Grant rule: if $g \in ?$, the take rule produces another graph with a transitive edge *a* $\subseteq$ *ß added.*



3. Create rule:



4. Remove rule:

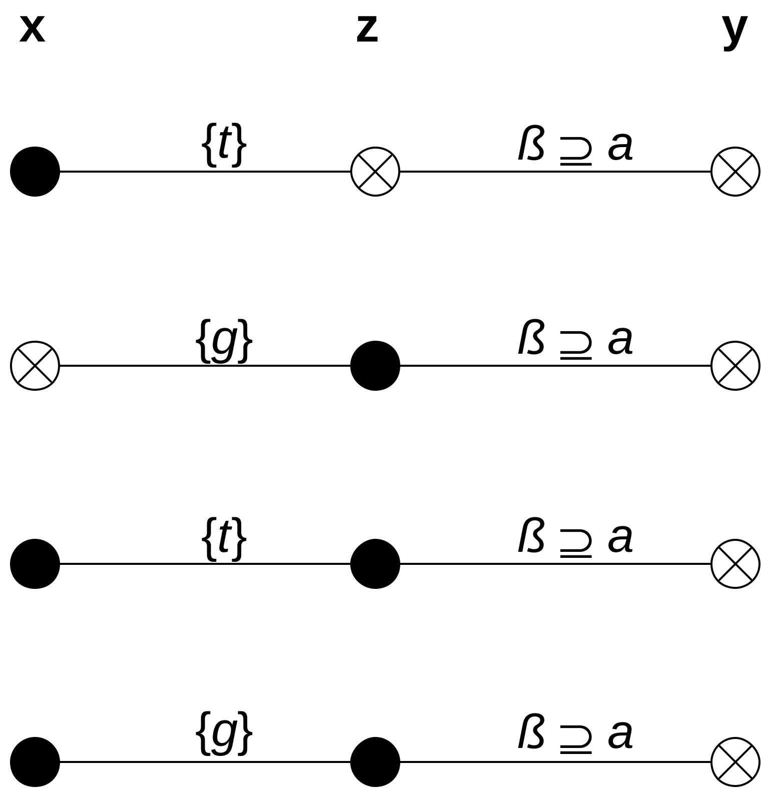


Take-Grant Protection Model: Sharing

- Given G_0, can vertex **x** obtain a rights over **y**?
  - Can_share(a,x, y,G_0) is true iff
    - $G_0 \vdash^* G_n$ using the four rules, &
    - There is an a edge from x to y in G_n
- *tg-path*: $\mathbf{v}_0,\ldots,\mathbf{v}_n$ with t or g edge between any pair of vertices $\mathbf{v}_i$, $\mathbf{v}_{i+1}$
  - Vertices *tg-connected* if *tg-path* between them
- Theorem: Any two subjects with *tg-path* of length 1 can share rights

Any two subjects with *tg-path* of length 1 can share rights

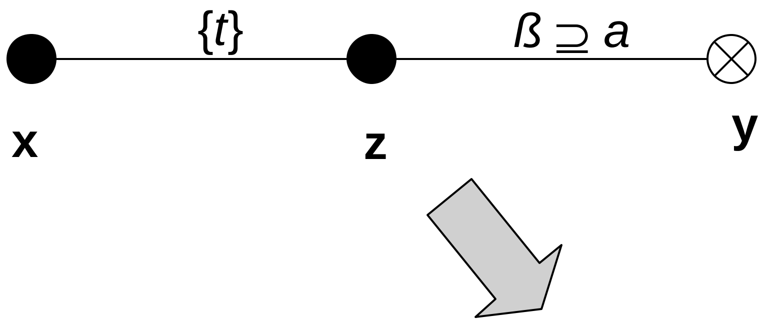
Can_share(a, ***x***, ***y***, G_0)



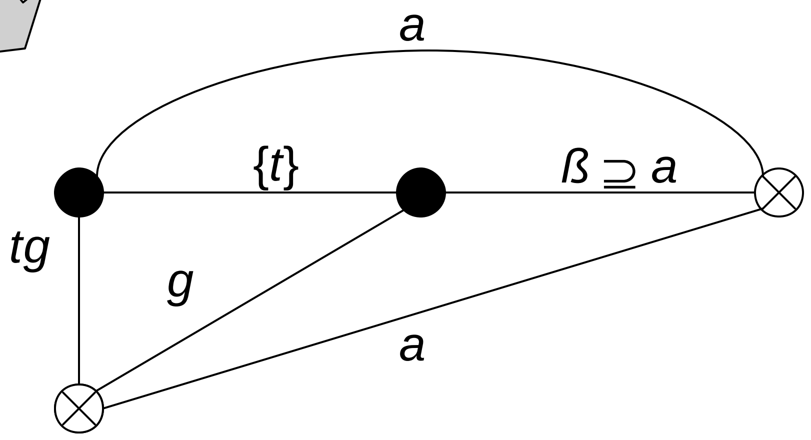
- Four possible length 1 *tg*-paths
  1. Take rule
  2. Grant rule
  3. Lemma 3.1
  4. Lemma 3.2

Any two subjects with *tg-path* of length 1 can share rights

Can_share(a, ***x***, ***y***,G_0)



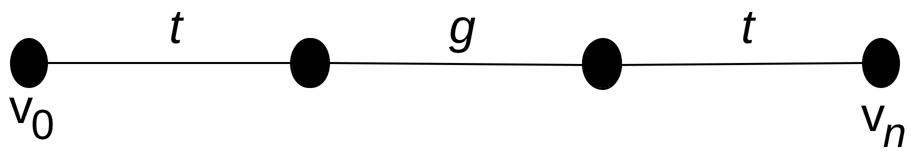
- Lemma 3.1
  - Sequence:
    - Create
    - Take
    - Grant
    - Take



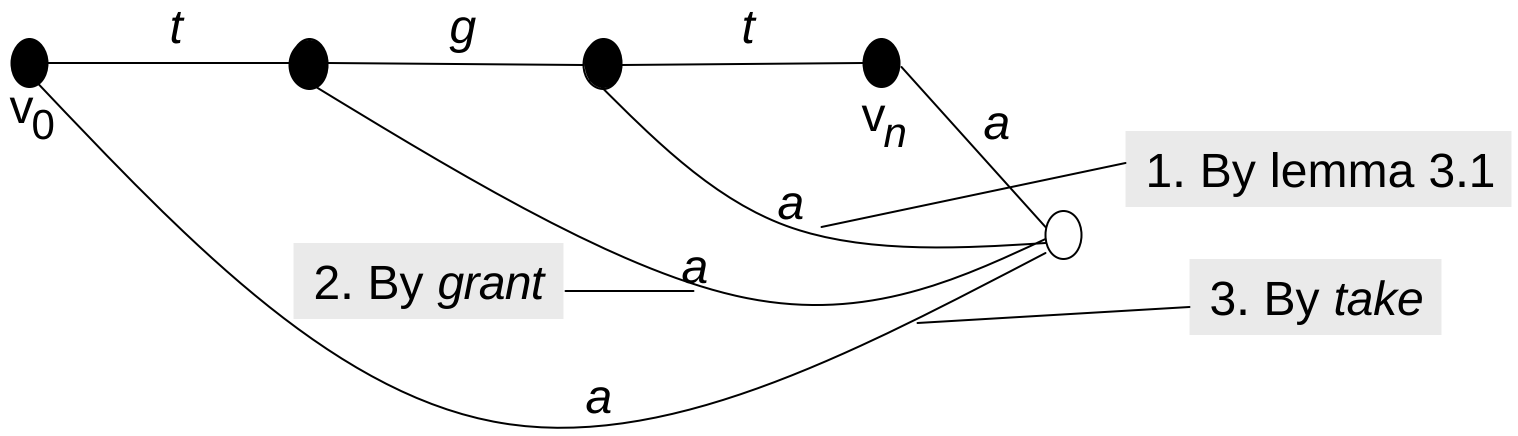
Other definitions

- Island: Maximal *tg*-connected subject-only subgraph
  - Can_share all rights in island
  - Proof: Induction from previous theorem
- Bridge: *tg*-path between subjects v_0 and v_n with edges of the following form:
  - $t_? *, t_? *$
  - $t_? *, g_? , t_? *$
  - $t_? *, g_? , t_? *$

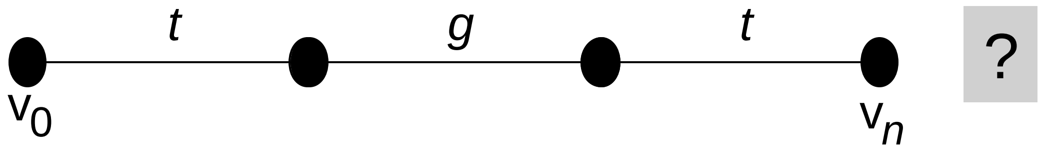
Bridge



t
g
t
v0
vn



t
g
t
v0
vn
a
a
a
a
1. By lemma 3.1
2. By grant
3. By take

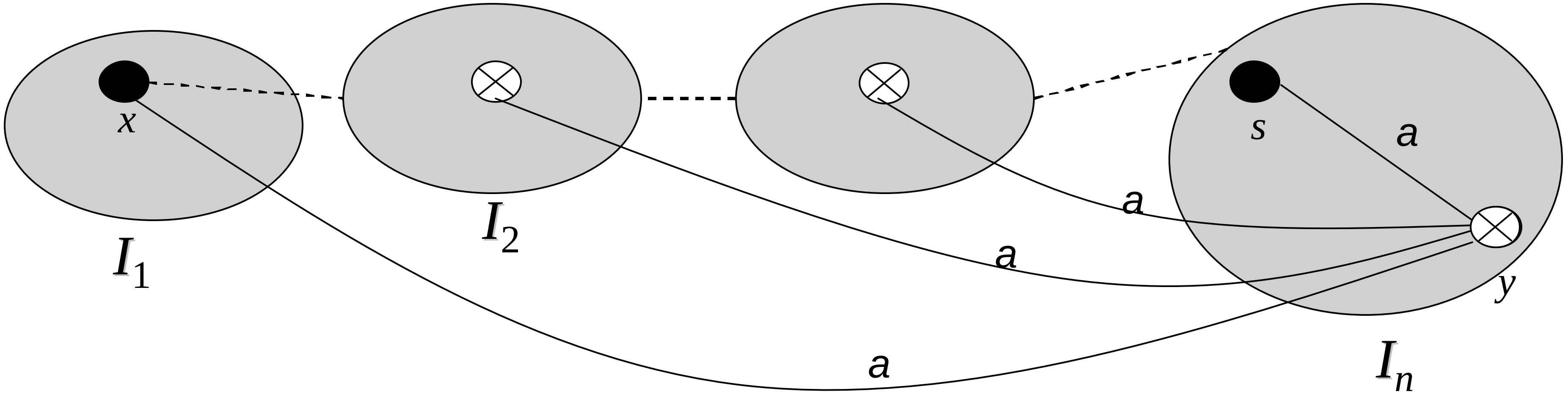


t
g
t
v0
vn
?

Theorem: Can_share(a,**x**,**y**,G_0) (for subjects)

- Subject_can_share(a, ***x***, ***y***,G_0) is true iff if ***x*** and ***y*** are subjects and
  - there is an a edge from x to y in G_0

  OR if:
  - $\exists$ a subject s $\in$ G_0 with an ***s***-to-***y*** a edge, 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}

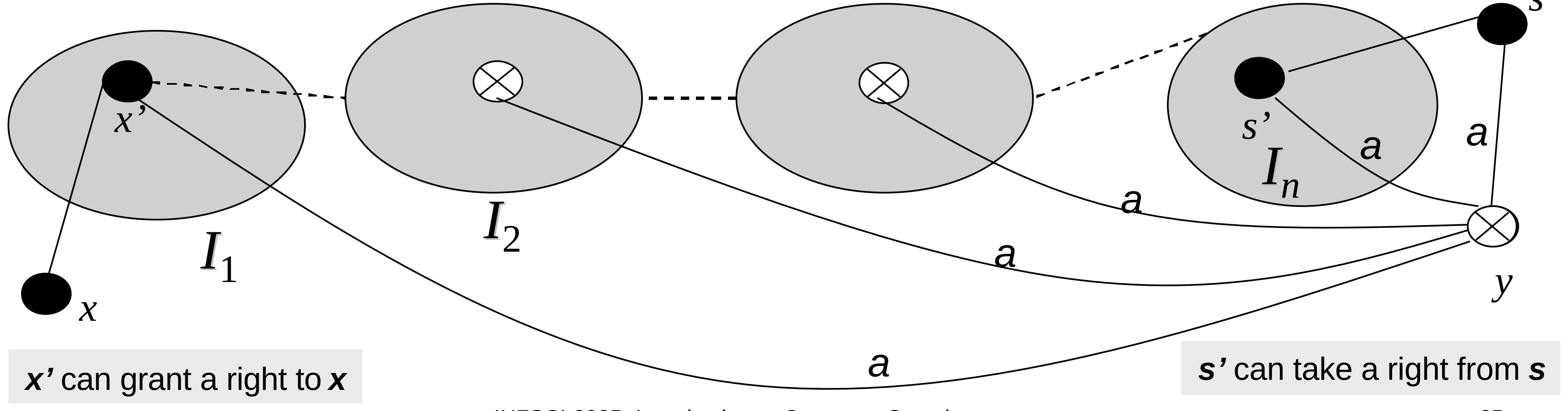


What about objects? Initial, terminal spans

- ***x*** *initially spans* to ***y*** if ***x*** is a subject and there is a *tg*-path associated with word {$t_?$ $*g_?$ } 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 {$t_?$ $*$} between them
  - ***x*** can take a right from ***y***

Theorem: Can_share(a,**x**,**y**,G_0)

- Can_share(a,x, y,G_0) iff there is an a edge from x to y in G_0 or if:
  - ∃ a vertex $s \in G_0$ with an s to y a edge,
  - ∃ a subject x' such that $x'=x$ or x' *initially spans* to x,
  - ∃ a subject s' such that $s'=s$ or s' *terminally spans* to s, and
  - ∃ 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}



s
x'
s'
a
a
a
a
a
I_1
I_2
I_n
x
y
x' can grant a right to ***x***
s' can take a right from ***s***

Theorem: Can_share(a,**x**,**y**,G_0)

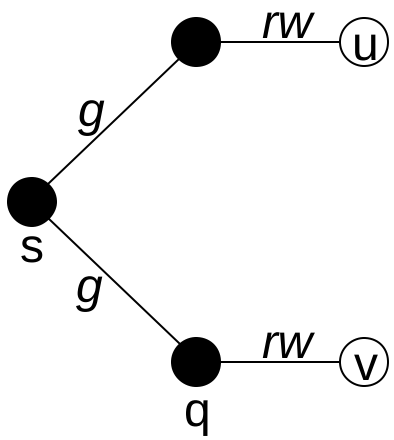
- Corollary: There is an $O(|V|+|E|)$ algorithm to test can_share: Decidable in linear time!!
- Theorem:
  - Let G_0 = ●, R a set of rights.
  - $G_0 \vdash^* G$ iff G is a finite directed acyclic graph, with edges labeled from R, and at least one subject with no incoming edge.
  - *Only if* part: v is initial subject and $G_0 \vdash^* G$;
    - No rule allows the deletion of a vertex
    - No rule allows the 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 and $G_0 \vdash^* G$
  - Assume v is the vertex with no incoming edge and apply rules

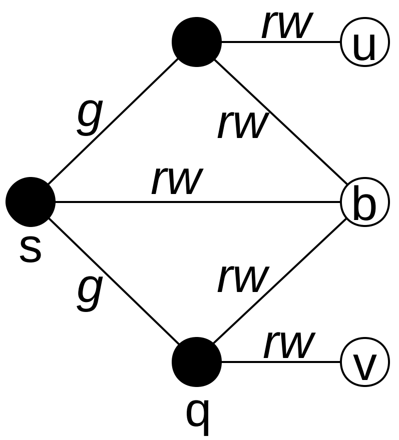
1. Perform "v creates ($\alpha \cup \{g\}$ to) new x_i" for all 2<=i <= n, and α is union of all labels on the incoming edges going into x_i in G
2. For all pairs x, y with x α over y in G, perform "v grants (α to y) to x"
3. If *ß* is the set of rights x has over y in G, perform "v removes ($\alpha \cup \{g\}$ - *ß*) to y"

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 third party (e.g. operating system) that controls *b*



Witness
- S creates ({r, w}, to new object) b
- S grants ({r, w}, b) to p
- S grants ({r, w}, b) to q



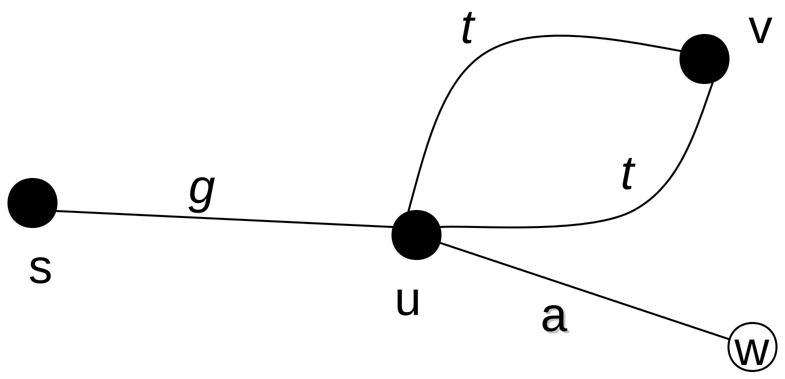
Theft in Take-Grant Model

- Can_steal(a,**x**,**y**,G_0) is true if there is no a edge from **x** to **y** in G_0 and ∃ sequence G_1, …, G_n s. t.:
  - ∃ a edge from **x** to **y** in $G_{n,}$,
  - ∃ rules $?_1$,…, $?_n$ that take $G_{i\text{-}1}$+ $?_n$ G_i , and
  - ∀ **v**,**w** ∈ G_i, 1=i<n, if ∃ a edge from **v** to **y** in G_0 then $?_i$ is not "**v** grants (a to **y**) to **w**"

- Disallows owners of a 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 (to w) from u



Theorem: When Theft Possible

- Can_steal(a,**x**,**y**,G_0) iff there is no a edge from **x** to **y** in G_0 and $\exists\, G_1, \ldots, G_n$ s. t.:
  - There is no a edge from **x** to **y** in G_0 ,
  - $\exists$ subject **x'** such that **x'**=**x** or **x'** *initially spans* to **x**, and
  - $\exists$ **s** with a edge to **y** in G_0 and can_share(*t*,**x'**,**s**,G_0)
- Proof:
  - $\Rightarrow$: Assume the three conditions hold
    - x can get t right over s (x is a subject)
    - x' creates a surrogate to pass to x (x is an object)
  - $\Leftarrow$: Assume can_steal is true:
    - No a edge from definition.
    - Can_share(a,**x**,**y**,G_0) from definition: a from **x** to **y** in G_n
    - **s** exists from can_share and earlier theorem
    - Can_share(*t*,**x'**,**s**,G_0): **s** can't grant a (definition), someone else must get a from **s**, show that this can only be accomplished with take rule

Conspiracy

- 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 *Can_share(a,**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

Conspiracy theorems:

- Deletion set d(**y**,**y’**): All **z** ∈ A(**y**) n A(**y’**) for which
  - **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’**
- 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 d(**y**,**y’**) not empty, edge from **y** to **y’**
- Theorem:

  Can_share(a,**x**,**y**,G_0) 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