Lecture #2

- Access control matrix
- Primitive operations and commands
- Miscellaneous points
- What is safety (security)?
- Is it decidable?

State Transitions

- Change the protection state of system
- $\vdash$ represents transition
  - $X_i \vdash_{\tau} X_{i+1}$: command τ moves system from state X_i to X_{i+1}
  - $X_i \vdash^{*} X_{i+1}$: a sequence of commands moves system from state X_i to X_{i+1}
- Commands often called *transformation procedures*

Primitive Operations

- **create subject** s; **create object** o
  - Creates new row, column in ACM; creates new column in ACM
- **destroy subject** s; **destroy object** o
  - Deletes row, column from ACM; deletes column from ACM
- **enter** r **into** $A[s, o]$
  - Adds r rights for subject s over object o
- **delete** r **from** $A[s, o]$
  - Removes r rights from subject s over object o

Create Subject

- Precondition: $s \notin S$
- Primitive command: **create subject** s
- Postconditions:
  - $S' = S \cup \{ s \}, O' = O \cup \{ s \}$
  - $(\forall y \in O')[a'[s, y] = \varnothing], (\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Create Object

- Precondition: $o \notin O$
- Primitive command: **create object** o
- Postconditions:
  - $S' = S, O' = O \cup \{ o \}$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Add Right

- Precondition: $s \in S$, $o \in O$
- Primitive command: enter r into $a[s, o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] \cup \{ r \}$
  - $(\forall x \in S')(\forall y \in O' - \{ o \})\ [a'[x, y] = a[x, y]]$
  - $(\forall x \in S' - \{ s \})(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Delete Right

- Precondition: $s \in S$, $o \in O$
- Primitive command: **delete** r **from** $a[s, o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] - \{ r \}$
  - $(\forall x \in S')(\forall y \in O' - \{ o \})\ [a'[x, y] = a[x, y]]$
  - $(\forall x \in S' - \{ s \})(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Destroy Subject

- Precondition: $s \in S$
- Primitive command: **destroy subject** s
- Postconditions:
  - $S' = S - \{ s \}, O' = O - \{ s \}$
  - $(\forall y \in O')[a'[s, y] = \varnothing], (\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S')(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Destroy Object

- Precondition: $o \in O$
- Primitive command: **destroy object** o
- Postconditions:
  - $S' = S$, $O' = O - \{ o \}$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S')(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Creating File

- Process *p* creates file *f* with *r* and *w* permission

```
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
```

Mono-Operational Commands

- Make process *p* the owner of file *g*

```
command make•owner(p, g)
    enter own into A[p, g];
end
```

- Mono-operational command
  - Single primitive operation in this command

Conditional Commands

- Let *p* give *q r* rights over *f*, if *p* owns *f*

```
command grant•read•file•1(p, f, q)
    if own in A[p, f]
    then
        enter r into A[q, f];
end
```

- Mono-conditional command
  - Single condition in this command

Multiple Conditions

- Let *p* give *q r* and *w* rights over *f*, if *p* has *r* and *c* rights over *q*

```
command grant•read•if•r•and•c(p, f, q)
    if r in A[p, q] and c in A[p, q]
    then
        enter r into A[q, f];
        enter w into A[q, f];
end
```

"Or" Conditions

- Let *p* give *q r* and *w* rights over *f*, if *p* has *r* or *c* rights over *q*

```
command grant•read•if•r(p, f, q)
    if r in A[p, f]
    then
        enter r into A[q, f];
        enter w into A[q, f];
end
```

“Or” Conditions

```
command grant•read•if•c(p, f, q)
    if c in A[p, f]
    then
        enter r into A[q, f];
        enter w into A[q, f];
end
command grant•read•if•r•or•c(p, f, q)
    grant•read•if•r(p, f, q);
    grant•read•if•c(p, f, q)
end
```

Copy Right

- Allows possessor to give rights to another
- Often attached to a right, so only applies to that right
  - *r* is read right that cannot be copied
  - *rc* is read right that can be copied
- Is copy flag copied when giving *r* rights?
  - Depends on model, instantiation of model

Own Right

- Usually allows possessor to change entries in ACM column
  – So owner of object can add, delete rights for others
  – May depend on what system allows
    - Can't give rights to specific (set of) users
    - Can't pass copy flag to specific (set of) users

Attenuation of Privilege

- Principle says you can't give rights you do not possess
  - Restricts addition of rights within a system
  - Usually *ignored* for owner
    - Why? Owner gives herself rights, gives them to others, deletes her rights.

What About Decidability?

- Give generic definition of “security”
- State decidability question in those terms
- Simple case: mono-operational commands
- General case: commands in general

What Is "Secure"?

- Adding a generic right *r* where there was not one is "leaking"
- If a system *S*, beginning in initial state s_0, cannot leak right *r*, it is *safe with respect to the right r*.

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*?
  - 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 **create**s 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

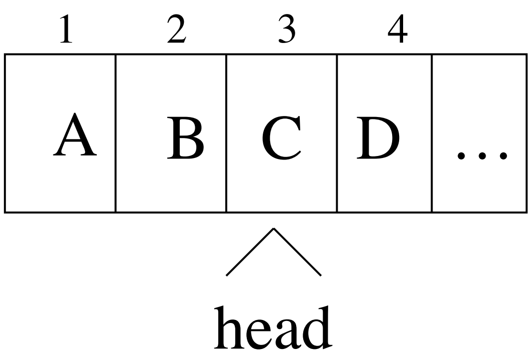
- Answer: *no*
- Sketch of proof:

  Reduce halting problem to safety problem

  Turing Machine review:

  – Infinite tape in one direction
  – States K, symbols M; distinguished blank b
  – Transition function $\delta(k, m) = (k', m', \mathrm{L})$ 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

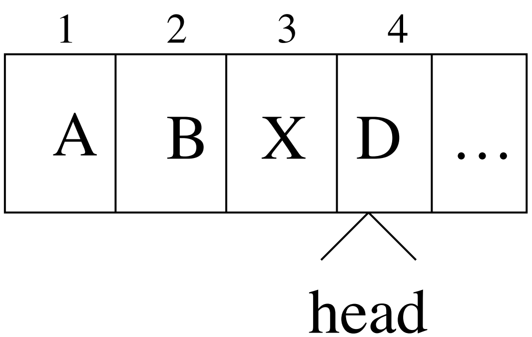
Mapping



Current state is k

	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	

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

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

command $\mathtt{c}_{k,\mathtt{C}}(s_3, s_4)$

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** $A[s_3, s_3]$;

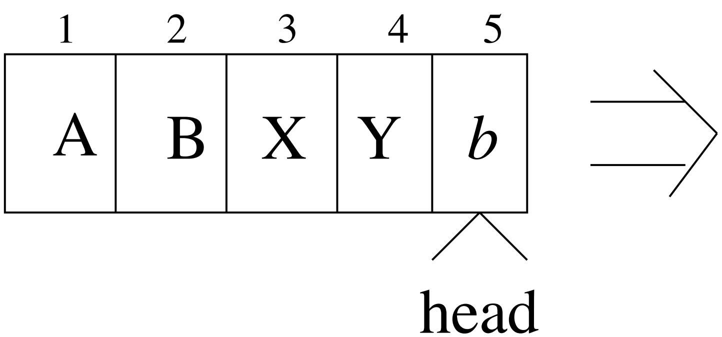
  delete C **from** $A[s_3, s_3]$;

  enter X **into** $A[s_3, s_3]$;

  enter k_1 **into** $A[s_4, s_4]$;

end

Mapping



	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

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

Command Mapping

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

```
command crightmost_{k,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

- 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 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)

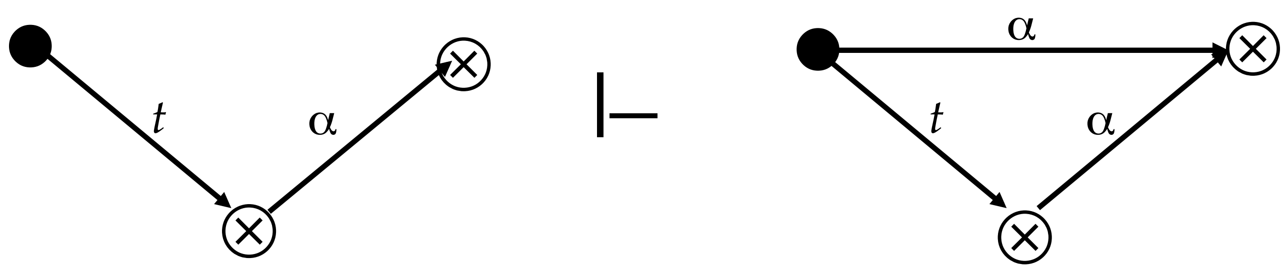
$G \vdash_x G'$ apply a rewriting rule x (witness) to G to get G'

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

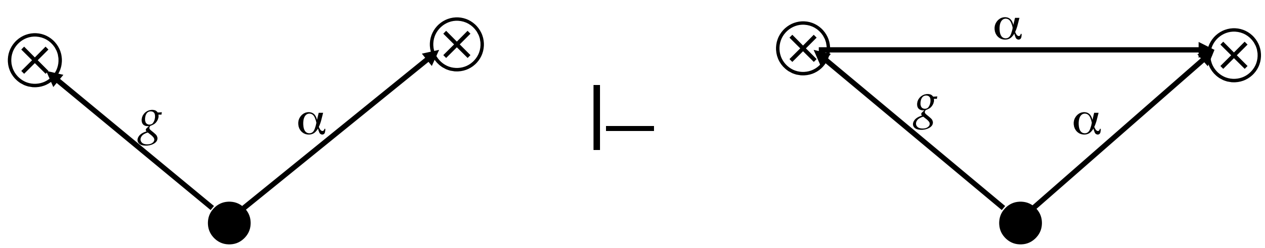
$R = \{ t, g, r, w, \ldots \}$ set of rights

Rules

take



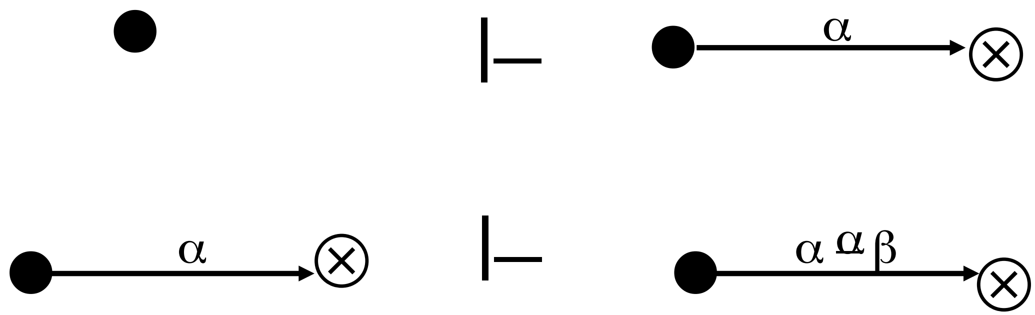
grant



More Rules

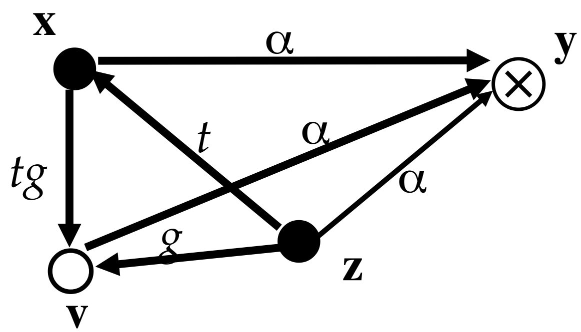
create

remove

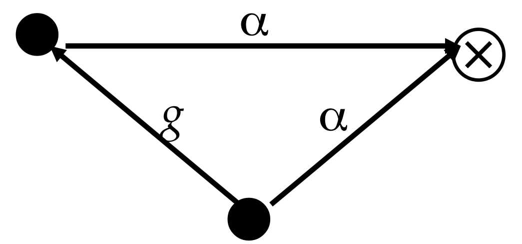


These four rules are called the *de jure* rules

Symmetry



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



Similar result for grant

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 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 $\{ \vec{t}^* \vec{g} \} \cup \{ \nu \}$
  - 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 $\{ \vec{t}^* \} \cup \{ \nu \}$
  - Means **x** can acquire any rights **y** has

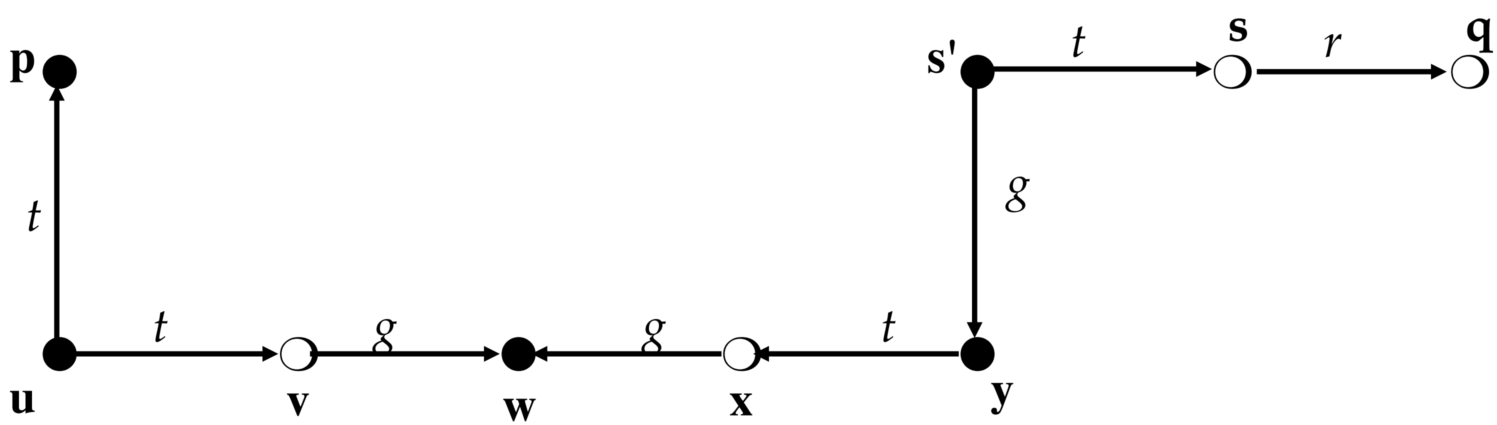
Bridges

- bridge: *tg*-path between subjects **x**, **y**, with associated word in

$$\{\ \overrightarrow{t}^*, \overrightarrow{t}^*, \overrightarrow{t}^*\overleftarrow{g}\ \overleftarrow{t}^*, \overrightarrow{t}^*\overrightarrow{g}\ \overleftarrow{t}^*\ \}$$

  - 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 **s's** (associated word $\vec{t}$)

can•share Predicate

Definition:

- *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 *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