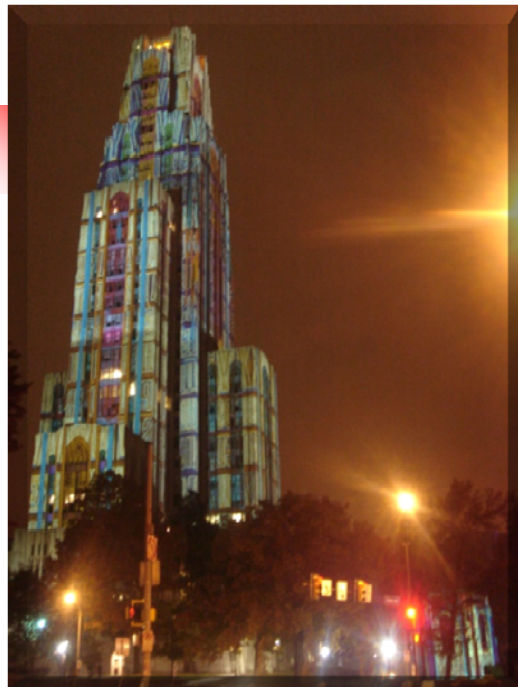


IS 2150 / TEL 2810

Information Security & Privacy



James Joshi
Associate Professor, SIS

Access Control Model
Foundational Results

Lecture 3
Jan 27, 2016



Objective

- Understand the basic results of the HRU model
 - Safety issue
 - Turing machine
 - Undecidability



Protection System

- State of a system
 - Current values of
 - memory locations, registers, secondary storage, etc.
 - other system components
- Protection state (P)
 - A system state that is considered secure
- A protection system
 - Captures the conditions for state transition
 - Consists of two parts:
 - A set of generic rights
 - A set of commands

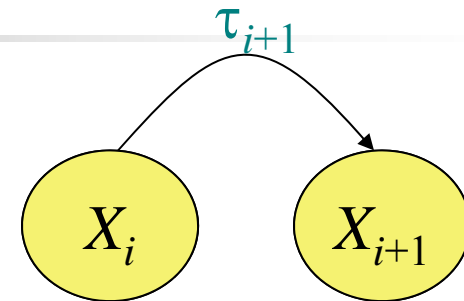


Protection System

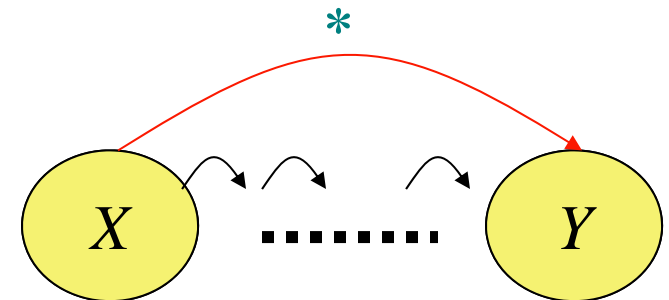
- Subject (S : set of all subjects)
 - Eg.: users, processes, agents, etc.
- Object (O : set of all objects)
 - Eg.: Processes, files, devices
- Right (R : set of all rights)
 - An action/operation that a subject is allowed/disallowed on objects
 - Access Matrix A : $a[s, o] \subseteq R$
- Set of Protection States: (S, O, A)
 - Initial state $X_0 = (S_0, O_0, A_0)$

State Transitions

$X_i \vdash_{\tau_{i+1}} X_{i+1}$: upon transition τ_{i+1} , the system moves from state X_i to X_{i+1}

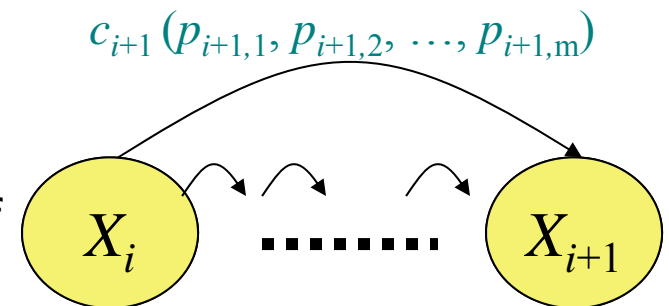


$X \vdash^* Y$: the system moves from state X to Y after a set of transitions



$X_i \vdash_{c_{i+1}(p_{i+1,1}, p_{i+1,2}, \dots, p_{i+1,m})} X_{i+1}$: 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; s does not exist prior to this
Create object o	Creates new column in ACM o does not exist prior to this
Enter r into $a[s, o]$	Adds r right for subject s over object o Ineffective if r is already there
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



Primitive commands (HRU)

Create subject s

Creates new row, column in ACM;
 s does not exist prior to this

Precondition: $s \notin S$

Postconditions:

$$S' = S \cup \{s\}, O' = O \cup \{s\}$$

$$(\forall y \in O')[a'[s, y] = \emptyset] \text{ (row entries for } s)$$

$$(\forall x \in S')[a'[x, s] = \emptyset] \text{ (column entries for } s)$$

$$(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$$



Primitive commands (HRU)

Enter r into $a[s, o]$ Adds r right for subject s over object o
Ineffective if r is already there

Precondition: $s \in S, o \in O$

Postconditions:

$$S' = S, O' = O$$

$$a'[s, o] = a[s, o] \cup \{ r \}$$

$$(\forall x \in S')(\forall y \in O')$$

$$[(x, y) \neq (s, o) \rightarrow a'[x, y] = a[x, y]]$$



System commands

- [Unix] 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



System commands

- Process p creates a new process q

Command $spawn_process(p, q)$

Create subject q ;

Enter own into $a[p, q]$

Enter r into $a[p, q]$

Enter w into $a[p, q]$

Enter r into $a[q, p]$

Enter w into $a[q, p]$

End

← Parent and child can
signal each other



System commands

- 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



Conditional Commands

- 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”??

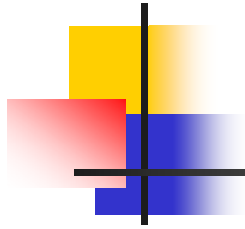
Command *grant_read_file*(*p*, *f*, *q*)
If *r* in *a*[*p*,*f*] OR *c* in *a*[*p*,*f*]
Then
Enter *r* into *a*[*q*,*f*]
End

Command *grant_read_file1*(*p*, *f*, *q*)
If *r* in *a*[*p*,*f*]
Then
Enter *r* into *a*[*q*,*f*]
End

Command *grant_read_file2*(*p*, *f*, *q*)
If *c* in *a*[*p*,*f*]
Then
Enter *r* into *a*[*q*,*f*]
End



Executing command:
grant_read_file
is equivalent to executing
commands:
***grant_read_file1*;**
grant_read_file2



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
 - **Leakage of rights:** (unsafe with respect to right r)
 - Assume that A representing a secure state does not contain a right r in an element of A .
 - A right r is said to be leaked, if a sequence of operations/commands adds r to an element of A , which did not contain r



What is a secure system?

- 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?



Undecidable Problems

- Decidable Problem
 - A decision problem can be solved by an algorithm that halts on all inputs in a finite number of steps.
- Undecidable Problem
 - A problem that cannot be solved for all cases by any algorithm whatsoever

Decidability Results

(Harrison, Ruzzo, Ullman)

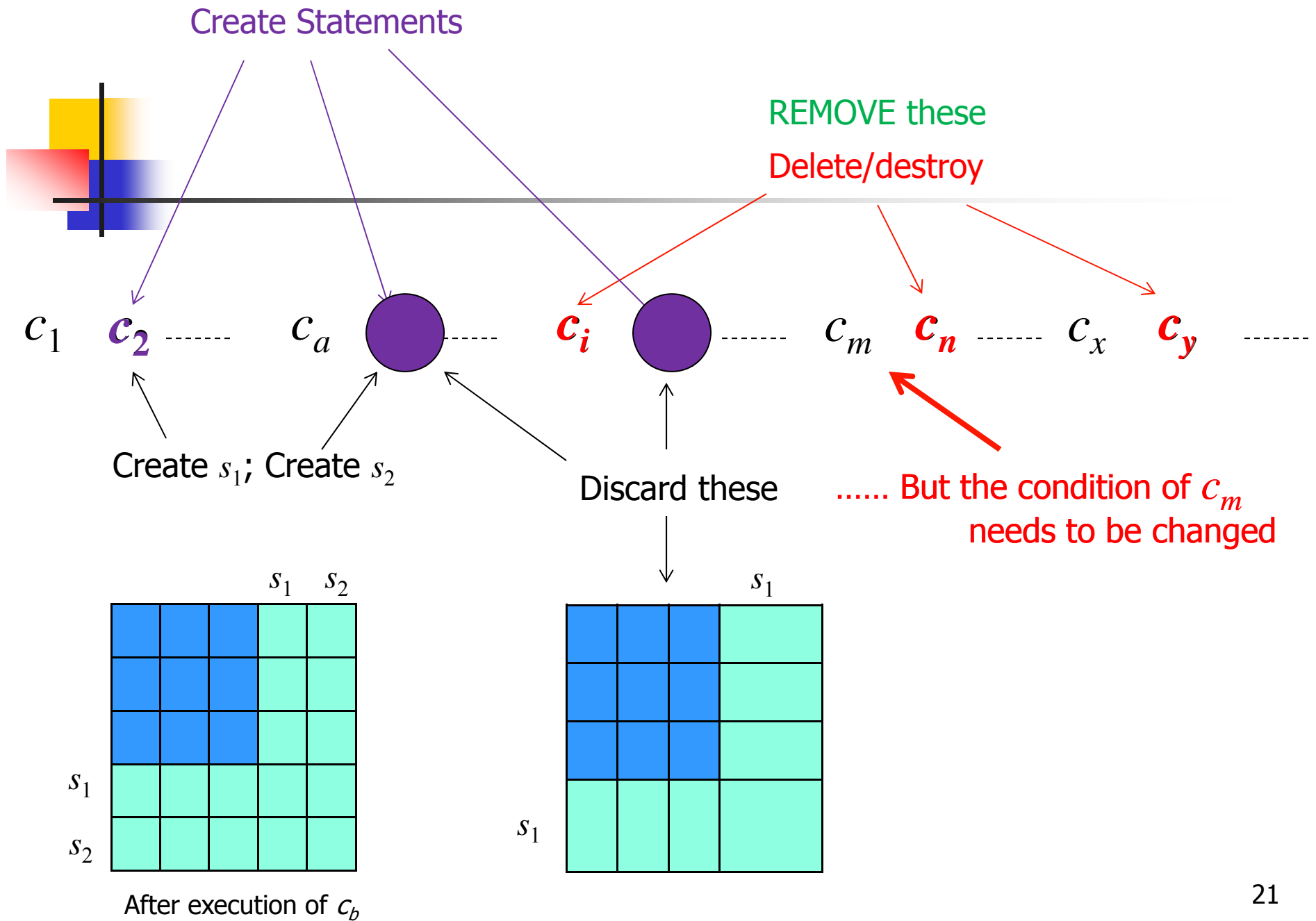
- Theorem:

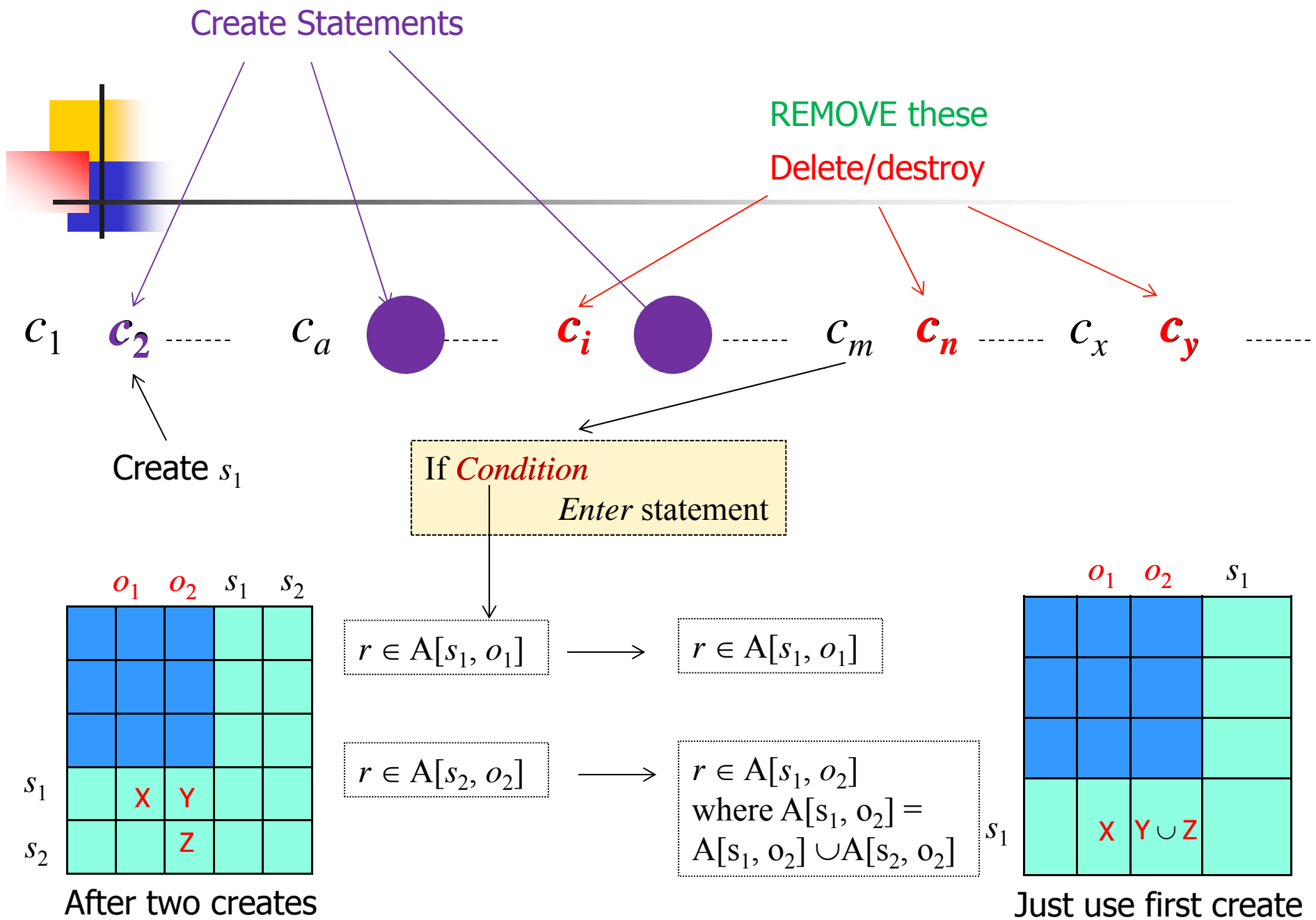
- Given a system where each command consists of a single *primitive* command (mono-operational), there exists an algorithm that will determine if a protection system with initial state X_0 is safe with respect to right r .

Decidability Results

(Harrison, Ruzzo, Ullman)

- Proof: determine minimum commands k to leak
 - Delete/destroy: Can't leak
 - Create/enter: new subjects/objects "equal", so treat all new subjects as one
 - No test for absence of right
 - Tests on $A[s_1, o_1]$ and $A[s_2, o_2]$ have same result as the same tests on $A[s_1, o_1]$ and $A[s_1, o_2] = A[s_1, o_2] \cup A[s_2, o_2]$
 - If n rights leak possible, must be able to leak $k = n(|S_o| + 1)(|O_o| + 1) + 1$ commands
 - Enumerate all possible states to decide





Decidability Results

(Harrison, Ruzzo, Ullman)

- Proof: determine minimum commands k to leak
 - Delete/destroy: Can't leak
 - Create/enter: new subjects/objects "equal", so treat all new subjects as one
 - No test for absence of right
 - Tests on $A[s_1, o_1]$ and $A[s_2, o_2]$ have same result as the same tests on $A[s_1, o_1]$ and $A[s_1, o_2] = A[s_1, o_2] \cup A[s_2, o_2]$
 - If n rights leak possible, must be able to leak $k = n(|S_o| + 1)(|O_o| + 1) + 1$ commands
 - Enumerate all possible states to decide



Decidability Results

(Harrison, Ruzzo, Ullman)

- It is undecidable if a given state of a given protection system is safe for a given generic right
- For proof – need to know Turing machines and halting problem



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).

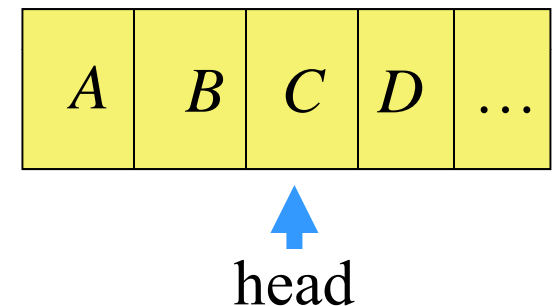


Turing Machine & Safety problem

- Theorem:
 - It is undecidable if a given state of a given protection system is safe for a given generic right
- 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)
- TM is an abstract model of computer
 - Alan Turing in 1936

Turing Machine

- 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 how to transition
 - What symbol to write
 - How to move the head ('L' for left and 'R' for right)
 - What is the next state

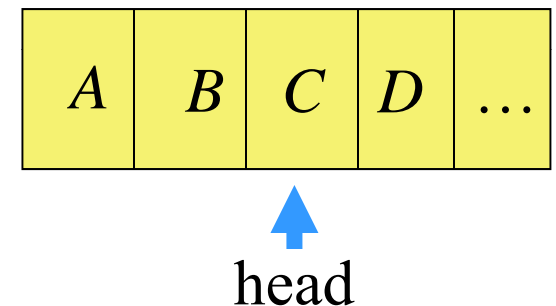


Current state is k

Current symbol is C

Turing Machine

- Transition function $\delta(k, m) = (k', m', L)$:
 - In state k , symbol m on tape location is replaced by symbol m' ,
 - Head moves one cell to the left, and TM enters state k'
- Halting state is q_f
 - TM halts when it enters this state

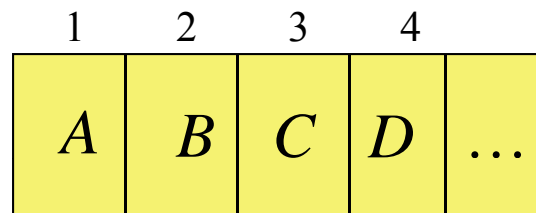


Current state is k

Current symbol is C

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

Turing Machine

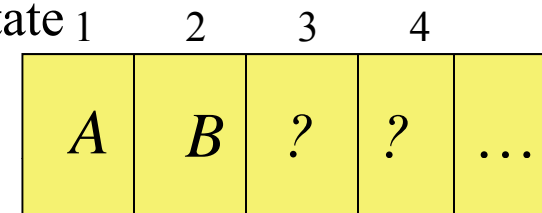
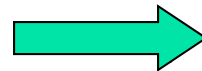


↑
head

Current state is k
Current symbol is C

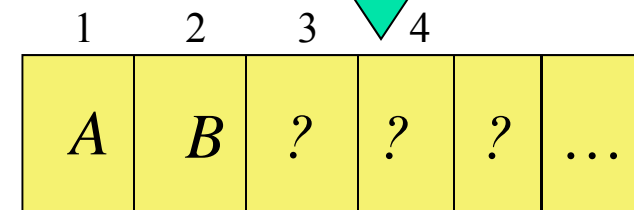
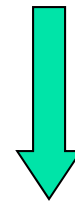
Let $\delta(k, C) = (k_1, X, R)$

where k_1 is the next state



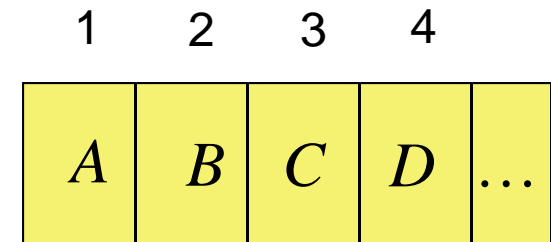
↑
head

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



↑
?
head

TM2Safety Reduction



Current state is k

Current symbol is C

↑
head

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 $1 \leq i < k$

	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	

Command Mapping (Left move)

1	2	3	4	
A	B	C	D	...

Current state is k

Current symbol is C



head

$$\delta(k, C) = (k_1, X, L)$$

$$\delta(k, C) = (k_1, X, L)$$

If head is not in leftmost

command $c_{k,C}(s_i, s_{i-1})$
 if own in $a[s_{i-1}, s_i]$ 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

	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	

Command Mapping (Left move)

1	2	3	4	
A	B	X	D	...

Current state is k_1

Current symbol is D head

$$\delta(k, C) = (k_1, X, L)$$

$$\delta(k, C) = (k_1, X, L)$$

If head is not in leftmost

command $c_{k,C}(s_i, s_{i-1})$
 if own in $a[s_{i-1}, s_i]$ 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

If head is in leftmost both s_i and s_{i-1} are s_1

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

Command Mapping (Right move)

1	2	3	4	
A	B	C	D	...

Current state is k

Current symbol is C



head

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

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

```

command  $c_{k,C}(s_i, s_{i+1})$ 
if  $own$  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
    
```

	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	

Command Mapping (Right move)

1	2	3	4	
A	B	X	D	...

Current state is k_1

Current symbol is C



head

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

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

command $c_{k,C}(s_i, s_{i+1})$
 if own 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

	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	

Command Mapping (Rightmost move)

1	2	3	4	
A	B	X	D	...

Current state is k_1

Current symbol is C

↑
head

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

$$\delta(k_1, C) = (k_2, Y, R)$$

command $\text{crightmost}_{k,C}(s_i, s_{i+1})$
 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 $a[s_i, s_i]$;

enter Y into $a[s_i, s_i]$;

enter k_2 into $a[s_i, s_i]$;

end

	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	

Command Mapping (Rightmost move)

1	2	3	4		...
A	B	X	Y		

Current state is k_1

Current symbol is D



head

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

$\delta(k_1, D) = (k_2, Y, R)$

command $\text{crightmost}_{k,C}(s_i, s_{i+1})$
 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 $a[s_i, s_i]$;
 enter Y into $a[s_i, s_i]$;
 enter k_2 into $a[s_i, s_i]$;

end

	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



Rest of Proof

- Protection system exactly simulates a TM
 - Exactly 1 *end* right in ACM
 - Only 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 results

- 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
- 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.



Summary

- HRU Model
- Some foundational results showing that guaranteeing security is hard problem