

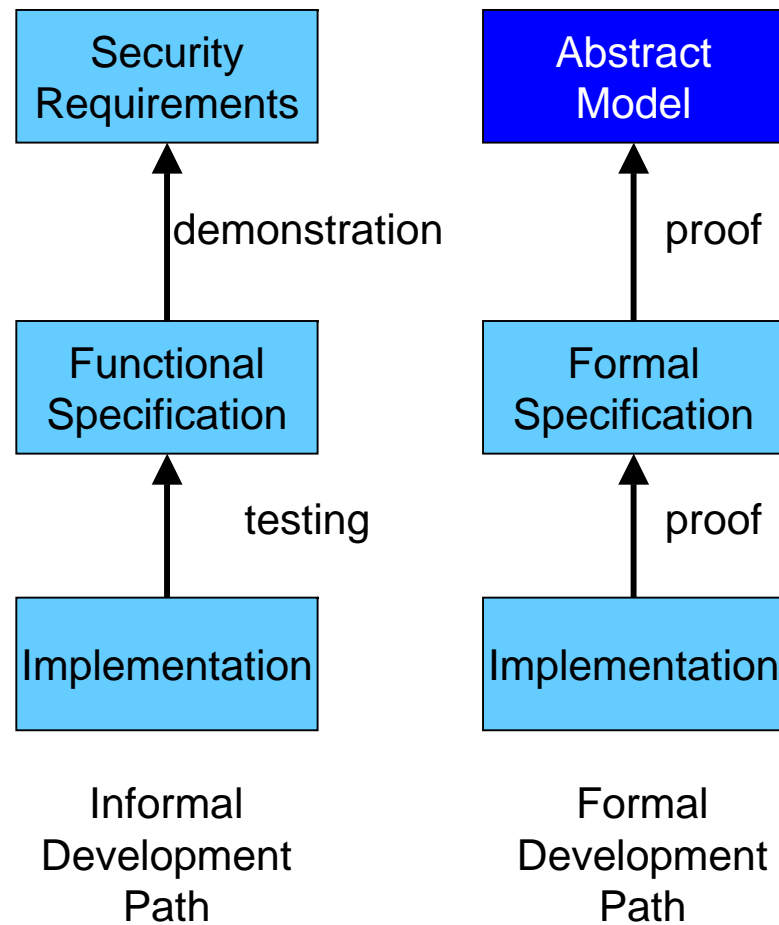
NIS/CpE 691A

Information System Security

**Stevens Institute of Technology
Spring 2006**

Class 4 – 2/9/06

From Last Time: Trustworthiness/Design Correctness



Security Model

- Creates the context for security requirements
- Properties:
 - Precise and unambiguous - or how would you design from it?
 - Rigorous implementation requires mathematical description
 - Natural language description
 - Simple and abstract
 - Easy to comprehend
 - Captures only *security*-related functions
 - Generic - does not constrain the implementation
 - Obvious representation of security policy

Who needs a security model?

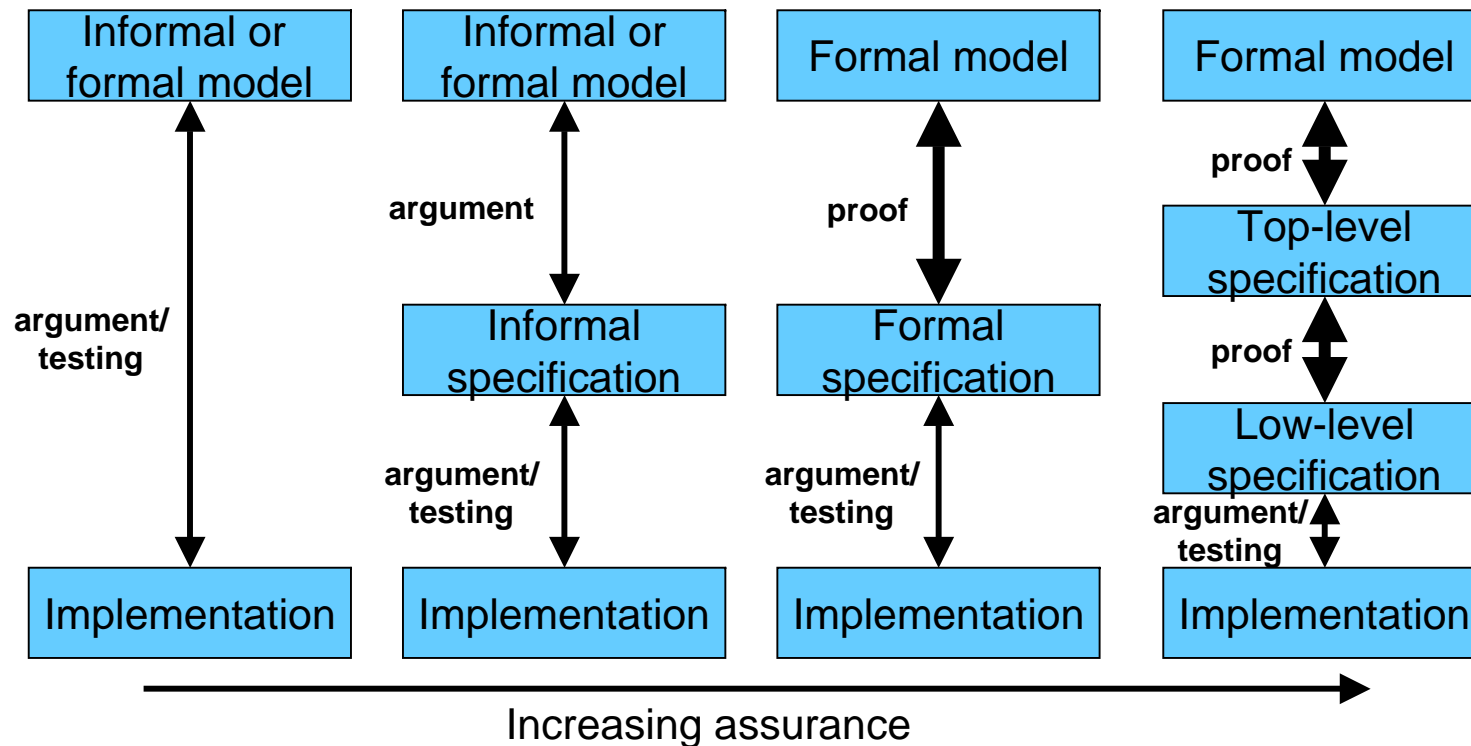
- What creates insecurity in a system design?
 - A bug in implementation, or
 - “What did you mean by security?”

Who needs a security model?

- What creates insecurity in a system design?
 - A bug in implementation, or
This area is addressed by careful system design
 - “What did you mean by security?”
This is the role of the security model

Who needs a security model?

- What creates insecurity in a system design?
 - A bug in implementation, or
This area is addressed by careful system design
 - “What did you mean by security?”
This is the role of the security model
- Security model creates a traceable record of system design



Types of Security Models

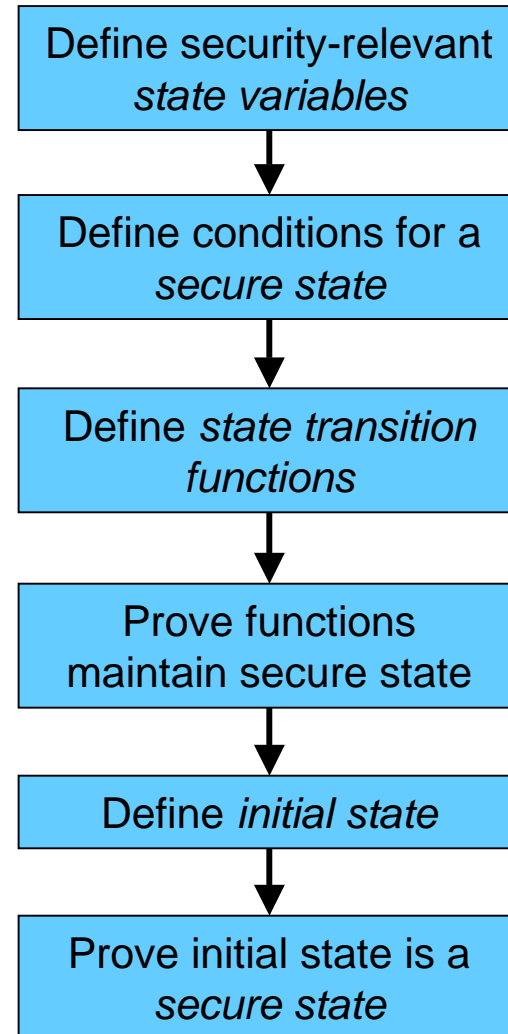
- State-machine model
 - System as an abstract mathematical state machine
 - State variables to represent state of machine at all times
 - Transition functions (rules of operation) define how state variables change
- Access matrix model
 - (State-machine model) + (subject-subject access rules)
- Access models
 - Security state of system defined by comparing subjects' security attributes
- Information flow model
 - (Access model) + (controls on information flows)
 - More useful than access models for covert channel analysis
- Non-interference model
 - Subjects in different domains are prevented from affecting each other if it violates systems security policies.

Characteristics of a Security Model

- A model is *NOT* a description or a specification
 - It *IS* an abstraction of them
 - A highly detailed model (a specification) encourages complexity that hides the high-level intent
- Easy to comprehend, therefore
 - brief
 - concise
 - unambiguous
- A restatement of the (natural language) security properties desired
- Attempts to be complete and consistent (but in the end, it can't really be)

State-Machine Security Models

- Mimics the actual execution of the system functions
- State variable = abstraction of every data variable present in system
- State transition functions = abstractions of system calls, object interactions



Example of State-Machine Model

- Policy: *A person may read a document only if the person's clearance is greater than or equal to the classification of the document*

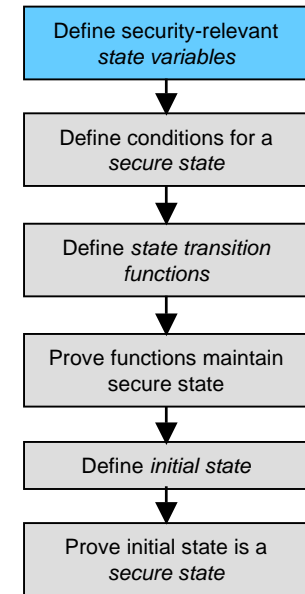
Real-world Item	Model Abstraction
person	subject
document	object
clearance	access class
classification	access class

- Translated property: #1: *A subject may read an object only if the access class of the subject is greater than or equal to the access class of the object*
- Additional property needed to meet intent of policy: #2: *A subject may write an object only if the access class of the object is greater than or equal to the access class of the subject.*

State-Machine Model

S = set of current subjects
 O = set of current objects
 $sclass(s)$ = access class of subject s
 $oclass(o)$ = access class of object o
 $A(s,o)$ = set of modes, equal to one of:
 $\{r\}$ if subject s can read object o
 $\{w\}$ if subject s can write object o
 $\{r,w\}$ if subject s can both read and write o
 ϕ if subject s can neither read nor write o
 $contents(o)$ = contents of object o
 $subj$ = active subject

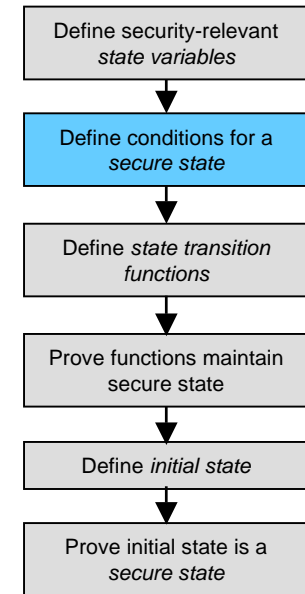
State of system at any one time is
 $\{S, O, sclass, oclass, A, contents, subj\}$



State-Machine Model

The definition of a secure state:
Mathematical translation of property #1 and property #2
into an invariant (independent of state):

Invariant: The system Σ is secure iff $\forall s \in S, o \in O,$
 $r \in A(s, o) \rightarrow sclass(s) \geq oclass(o),$
 $w \in A(s, o) \rightarrow oclass(o) \geq sclass(s)$

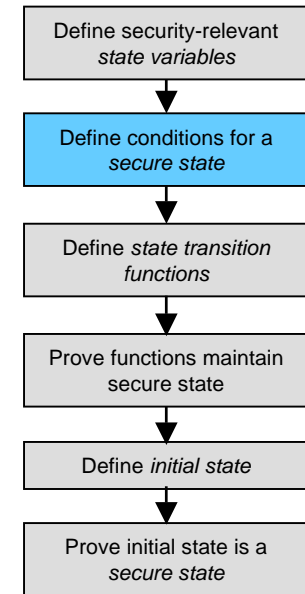


State-Machine Model

The definition of a secure state:
Mathematical translation of property #1 and property #2
into an invariant (independent of state):

Invariant: The system Σ is secure iff $\forall s \in S, o \in O,$
 $r \in A(s, o) \rightarrow sclass(s) \geq oclass(o),$
 $w \in A(s, o) \rightarrow oclass(o) \geq sclass(s)$

What if $A(s,o)$ were empty?

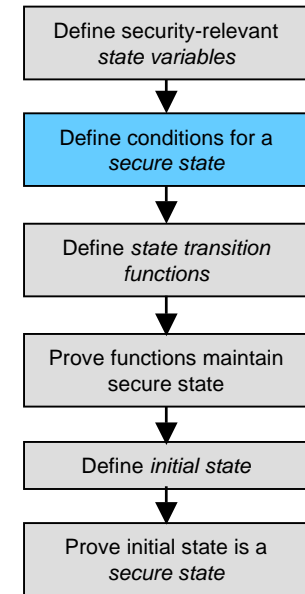


State-Machine Model

The definition of a secure state:
Mathematical translation of property #1 and property #2
into an invariant (independent of state):

Invariant: The system Σ is secure iff $\forall s \in S, o \in O,$
 $r \in A(s, o) \rightarrow sclass(s) \geq oclass(o),$
 $w \in A(s, o) \rightarrow oclass(o) \geq sclass(s)$

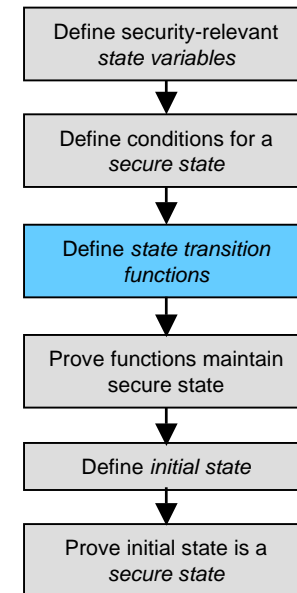
What if $A(s,o)$ were empty?
– Not an issue, the properties do not imply usefulness



State-Machine Model

Transition functions examples:

Create_object(<i>o,c</i>)	Create object <i>o</i> at class <i>c</i>
Set_access(<i>s,o,modes</i>)	Set access <i>modes</i> for subject <i>s</i> to object <i>o</i>
Create/Change_object(<i>o,c</i>)	Set class of <i>o</i> to <i>c</i> and create
Write_object(<i>o,d</i>)	Write data <i>d</i> into <i>contents(o)</i>
Copy_object(<i>from,to</i>)	Copy <i>contents(from)</i> to <i>contents(to)</i>
Append_data(<i>o,d</i>)	Add data <i>d</i> to <i>contents(o)</i>



State-Machine Model

Transition functions definitions:

Function 1: *Create_object(o, c)*

if $o \notin O$

then $O' = O \cup \{o\}$

$oclass'(o) = c$

$\forall s \in S, A'(s, o) = \emptyset$

Function 2: *Set_access(s, o, modes)*

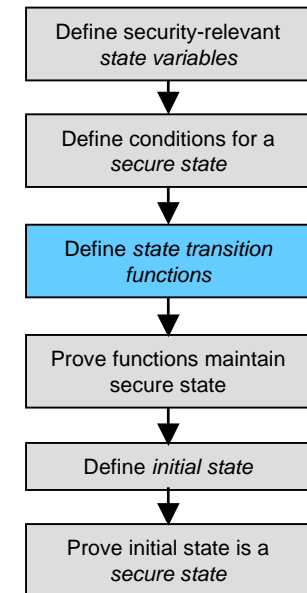
if $s \in S$ and $o \in O$

and if({ $[r \in modes \text{ and } sclass(s) \geq oclass(o)]$ or $r \notin modes$ })

and

{ $[w \in modes \text{ and } oclass(o) \geq sclass(s)]$ or $w \notin modes$ })

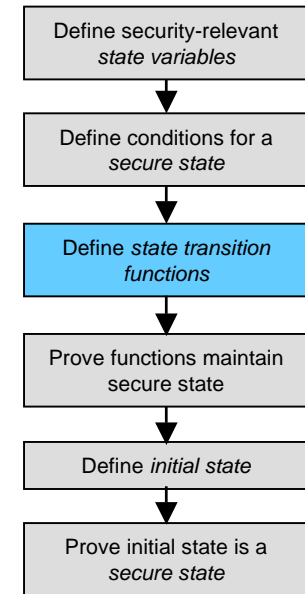
then $A'(s, o) = modes$



State-Machine Model

Transition functions definitions - comments:

- O' is the next state of O
- “=” is mathematical equality, not assignment
- There is no ordering of operations, all must be performed without interruption
- Operations are atomic, there is no passage of time between operations
- The function describes all allowed state transitions.



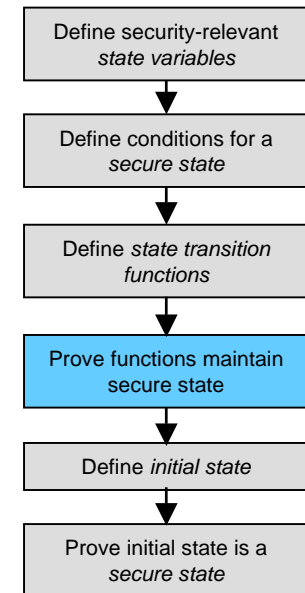
State-Machine Model

- Proof of security:
 - For each function, prove:

Invariant and **Function** imply **Invariant**'

“A system starting in a secure state is left in a secure state after application of Function”

- Proofs are left as an exercise for the student. (HW4)



State-Machine Model

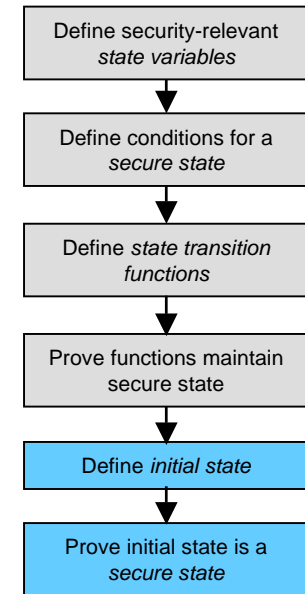
- Define and prove initial state security:

- Initial state:

$\{S_0, O_0, sclass_0, oclass_0, contents_0, subj_0\}$

- One possible initial state:

Initial State (1): $S_0 = \emptyset, O_0 = \emptyset$



State-Machine Model

- Define and prove initial state security:

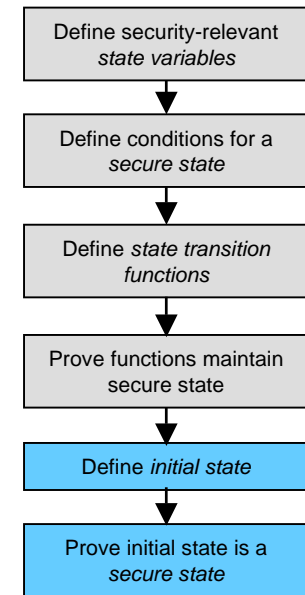
- Initial state:

$\{S_0, O_0, sclass_0, oclass_0, contents_0, subj_0\}$

- One possible initial state:

Initial State (1): $S_0 = \emptyset, O_0 = \emptyset$

(remember my comment in Class 1 on the tradeoff between security and performance? This system is as secure as a brick. No subjects, no objects, and no way to create one of those pesky subjects)



State-Machine Model

- Define and prove initial state security:

- Initial state:

$$\{S_0, O_0, sclass_0, oclass_0, contents_0, subj_0\}$$

- One possible initial state:

$$\text{Initial State (1): } S_0 = \emptyset, O_0 = \emptyset$$

(remember my comment in Class 1 on the tradeoff between security and performance? This system is as secure as a brick. No subjects, no objects, and no way to create one of those pesky subjects)

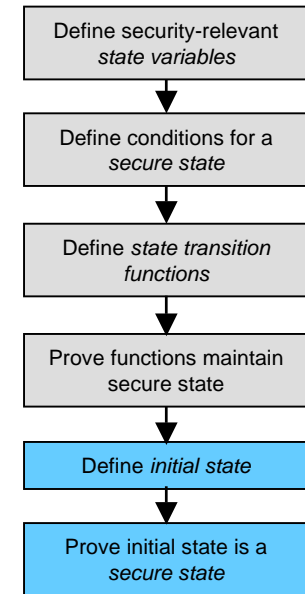
- A more interesting initial state:

$$\text{Initial State (2): } \forall s \in S, \forall o \in O$$

$$sclass_0(s) = c_0$$

$$oclass_0(o) = c_0$$

$$A_0(s, o) = \{r, w\}$$



State-Machine Model

- Define and prove initial state security:

- Initial state:

$$\{S_0, O_0, sclass_0, oclass_0, contents_0, subj_0\}$$

- One possible initial state:

$$\text{Initial State (1): } S_0 = \emptyset, O_0 = \emptyset$$

(remember my comment in Class 1 on the tradeoff between security and performance? This system is as secure as a brick. No subjects, no objects, and no way to create one of those pesky subjects)

- A more interesting initial state:

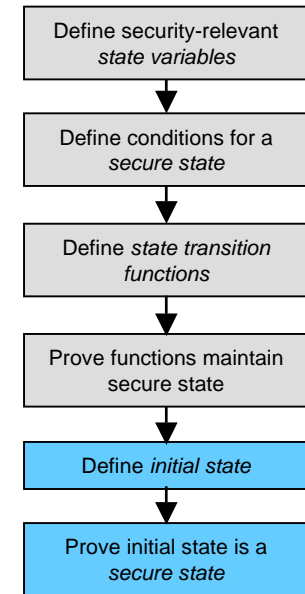
$$\text{Initial State (2): } \forall s \in S, \forall o \in O$$

$$sclass_0(s) = c_0$$

$$oclass_0(o) = c_0$$

$$A_0(s, o) = \{r, w\}$$

What is this equivalent to?



State-Machine Model

- Adding constraints to State-Machine Access Models
 - Security is more than the static view of security given by the Secure State Invariant - it must also allow a way to capture information about a sequence of states.
 - Constraints address:
 - Nonsecure transitions: old and new value of variables must maintain a “secure” relationship
 - Controls on subjects: subjects should not be allowed to invoke certain operations
 - Controls on information: if information content is referenced by model, it must control transitions that modify information.

State-Machine Model

- Nonsecure Transitions:
 - rewrite *Create_object*() to allow change in access class

Function 3: *Create / Change_object*(o, c)

$oclass'(o) = c$; and

if $o \notin O$

then $O' = O \cup \{o\}$

$oclass'(o) = c$

$\forall s \in S, A'(s, o) = \emptyset$

State-Machine Model

- Nonsecure Transitions:

- rewrite *Create_object()* to allow change in access class

Function 3: *Create / Change_object(o, c)*

$oclass'(o) = c;$ and

if $o \notin O$

then $O' = O \cup \{o\}$

$oclass'(o) = c$

$\forall s \in S, A'(s, o) = \emptyset$

- This creates a security violation, so add property:

Property #3: The access class of an object cannot decrease.

State-Machine Model

- Nonsecure Transitions:

- rewrite *Create_object()* to allow change in access class

Function 3: *Create / Change_object(o, c)*

$oclass'(o) = c;$ and

if $o \notin O$

then $O' = O \cup \{o\}$

$oclass'(o) = c$

$\forall s \in S, A'(s, o) = \emptyset$

- This creates a security violation, so add property:

Property #3: The access class of an object cannot decrease.

- *Create/Change_object()* is in violation of Property #3.

- Add constraint:

Constraint 1: $\forall o \in O, oclass'(o) \geq oclass(o)$

security classifications can be **upgraded** on objects

State-Machine Model

- Controls on Subjects
 - Subjects should not be allowed to give themselves access to an object

Property #4: A subject may modify another subject's access to an object only if the first subject can read the object

Constraint 2: $\forall o \in O,$
if $r \notin A(subj, o) \rightarrow \forall s \in S, A'(s, o) = A(s, o)$

State-Machine Model

- Controls on Information

- Only changes to access rights are defined, what about modeling operations on data contents?

Function 4: *Write_object*(o, d)

if $o \in O$ and $w \in A(\text{subj}, o)$

then $\text{contents}'(o) = d$

- *Write_object*() does not modify any variables in invariants or constraints, so it is secure according to model.

State-Machine Model

- Controls on Information

- Only changes to access rights are defined, what about modeling operations on data contents?

Function 4: $Write_object(o, d)$

if $o \in O$ and $w \in A(subj, o)$

then $contents'(o) = d$

- $Write_object()$ does not modify any variables in invariants or constraints, so it is secure according to model.
- The model (not the policy) is insufficient - it only talks about write access, not about the act of writing. We need to add something to deal with information in the model.

Property #5: A subject may modify an object only if the subject has write access to the object

State-Machine Model

- Controls on Information

- Only changes to access rights are defined, what about modeling operations on data contents?

Function 4: $Write_object(o, d)$

if $o \in O$ and $w \in A(subj, o)$

then $contents'(o) = d$

- $Write_object()$ does not modify any variables in invariants or constraints, so it is secure according to model.
- The model (not the policy) is insufficient - it only talks about write access, not about the act of writing. We need to add something to deal with information in the model.

Property #5: A subject may modify an object only if the subject has write access to the object

Constraint 3: $\forall o \in O,$

if $w \notin A(subj, o) \rightarrow contents'(o) = contents(o)$

State-Machine Model

- Comments on refining the model
 - Adding more detail to the model addresses other ways security might be compromised in the system
 - Detail (more state variables, more constraints) compounds the proof of correctness of the specification
 - Don't add trivially true constraints
 - Is the constraint security-relevant?
 - Does it express something that is an extension of the security policy?
 - Would leaving out the constraint create a security loophole?

State-Machine Model

- Bell and LaPadula Security Model
 - Targets the military security policy
 - For multilevel security
 - Establishes a partial ordering between access classes (Classification and Category Set)

State-Machine Model

- Bell and LaPadula Security Model

- Partial ordering between access classes:

- “Dominates” relationship: \geq Not a numerical comparison

- Partial ordering on sets:

Reflexive: $A \geq A$

Antisymmetric: if $A \geq B$ and $B \geq A$ then $A = B$

Transitive: if $A \geq B$ and $B \geq C$ then $A \geq C$

But if $A \not\geq B$, it is not true that $B \geq A$

FIX UNICODE AND ACROBAT

State-Machine Model

- Bell and LaPadula Security Model

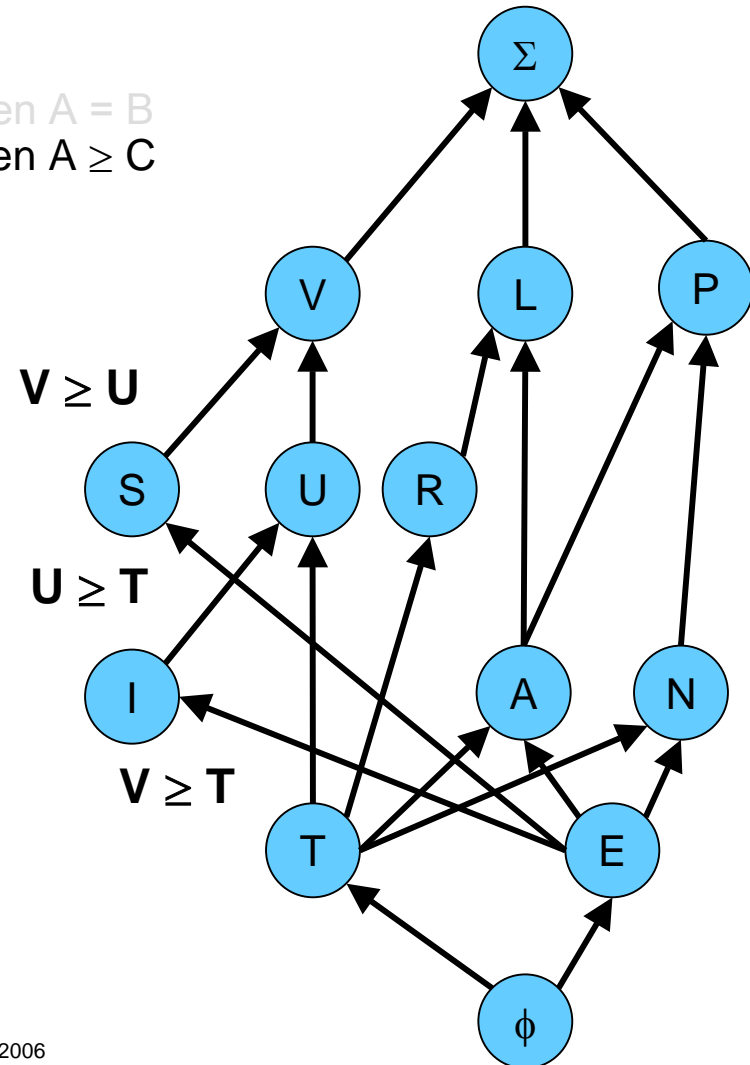
– Partial ordering on sets:

Reflexive: $A \geq A$

Antisymmetric: if $A \geq B$ and $B \geq A$ then $A = B$

Transitive: if $A \geq B$ and $B \geq C$ then $A \geq C$

But if $A \not\geq B$, it is not true that $B \geq A$



State-Machine Model

- Bell and LaPadula Security Model

– Partial ordering on sets:

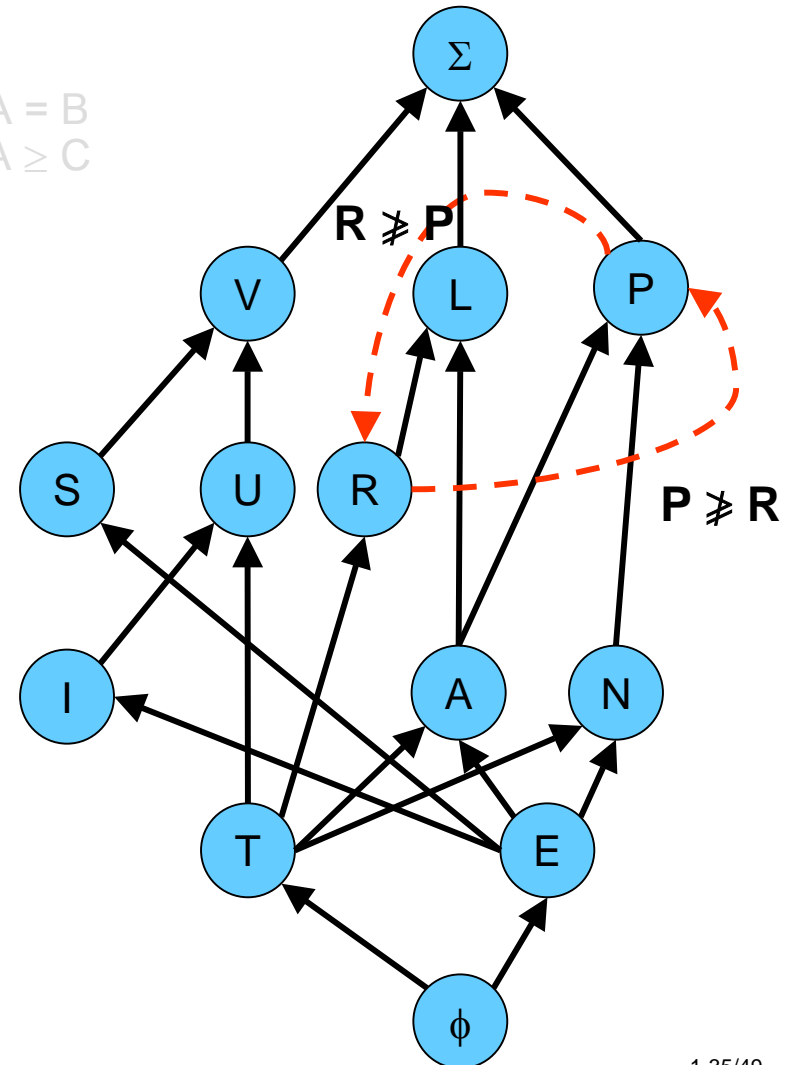
Reflexive: $A \geq A$

Antisymmetric: if $A \geq B$ and $B \geq A$ then $A = B$

Transitive: if $A \geq B$ and $B \geq C$ then $A \geq C$

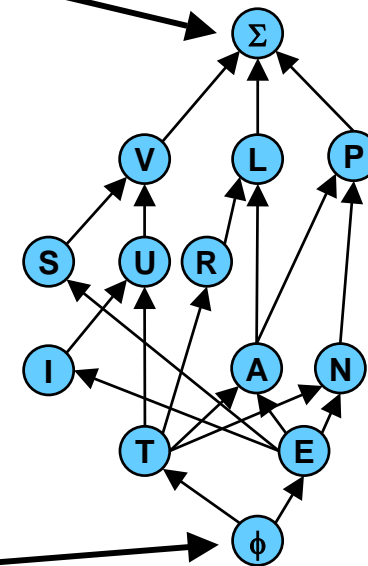
But if $A \not\geq B$, it is not true that $B \geq A$

There is no relationship between P and R



State-Machine Model

- The “Dominates” relationship defines a lattice:
 - In the set of all access classes that dominate both A and B, there is a unique *least upper bound* that is dominated by all the other upperbounds



- In the set of all access classes dominated by A and B, there is a unique *greatest lower bound* that dominates all the other lower bounds
- The fact that the Dominates operator defines a lattice makes it useful for other reasoning about the relationship

Mandatory vs. Discretionary Controls

- Mandatory access control:
 - Based on a rigidly enforced security policy: e.g., SECRET users cannot access TOP SECRET information
- Discretionary access control:
 - User specified controls: e.g., UNIX r-w-x permissions for self-group-all as set by a user on a per-file basis

Mandatory vs. Discretionary Controls

- Mandatory access control:
 - Based on a rigidly enforced security policy: e.g., SECRET users cannot access TOP SECRET information
- Discretionary access control:
 - User specified controls: e.g., UNIX r-w-x permissions for self-group-all as set by a user on a per-file basis
- How do you perform system backups/restores, user add, modify user security levels, etc.

Mandatory vs. Discretionary Controls and Trusted Subjects

- Mandatory access control:
 - Based on a rigidly enforced security policy: e.g., SECRET users cannot access TOP SECRET information
- Discretionary access control:
 - User specified controls: e.g., UNIX r-w-x permissions for self-group-all as set by a user on a per-file basis
- How do you perform system backups/restores, user add, modify user security levels, etc.
 - Trusted subjects are allowed to violate traditional security controls
 - confinement (*-property) (reading/writing information at different security levels)
 - tranquility (object classes are not allowed to change)

Information-Flow Models

- The Bell and LaPadula model, as well as other State-Machine models do not protect against covert channels.
- Thus, they don't insure security against (e.g.) the Trojan Horses that they were designed to deal with
- Consider:

Function 5: *Copy_object(from,to)*

if $from \in O$ and $to \in O$ and $w \in A(subj,to)$

then $contents'(to) = contents(from)$

What is wrong with this function?

Information-Flow Models

- The Bell and LaPadula model, as well as other State-Machine models do not protect against covert channels.
- Thus, they don't insure security against (e.g.) the Trojan Horses that they were designed to deal with
- Consider:

Function 5: *Copy_object(from,to)*

if $from \in O$ and $to \in O$ and $w \in A(subj,to)$

then $contents'(to) = contents(from)$

What is wrong with this function?

Does *subj* have read permissions on *from*?

Maybe he doesn't - he could later read *to*, where he might have direct read permission, compromising read access to an object (*from*) he should not have read

Information-Flow Models

- The Bell and LaPadula model, as well as other State-Machine models do not protect against covert channels.
- Thus, they don't insure security against (e.g.) the Trojan Horses that they were designed to deal with
- Consider:

Function 5: $Copy_object(from, to)$

if $from \in O$ and $to \in O$ and $w \in A(subj, to)$

then $contents'(to) = contents(from)$

What is wrong with this function?

Does $subj$ have read permissions on $from$?

Maybe he doesn't - he could later read to , where he might have direct read permission, compromising read access to an object ($from$) he should not have read

– Can you fix this with a constraint or invariant?

Information-Flow Models

- The Bell and LaPadula model, as well as other State-Machine models do not protect against covert channels.
- Thus, they don't insure security against (e.g.) the Trojan Horses that they were designed to deal with
- Consider:

Function 5: $Copy_object(from, to)$

if $from \in O$ and $to \in O$ and $w \in A(subj, to)$

then $contents'(to) = contents(from)$

What is wrong with this function?

Does $subj$ have read permissions on $from$?

Maybe he doesn't - he could later read to , where he might have direct read permission, compromising read access to an object ($from$) he should not have read

– Can you fix this with a constraint or invariant?

- Scan all functions for references to $contents(o)$ not qualified by $r \in A(subj, o)$

Information-Flow Models

- Can you fix this with a constraint or invariant?
 - Scan all functions for references to *contents(o)* not qualified by $r \in A(\text{subj}, o)$

Information-Flow Models

- Can you fix this with a constraint or invariant?
 - Scan all functions for references to $contents(o)$ not qualified by $r \in A(subj, o)$

What about:

Function 6: *Append _data(o, d)*

if $o \in O$ and $w \in A(subj, to)$

then $contents'(o) = contents(o) \cup \{d\}$

Information-Flow Models

- Can you fix this with a constraint or invariant?
 - Scan all functions for references to $contents(o)$ not qualified by $r \in A(subj, o)$

What about:

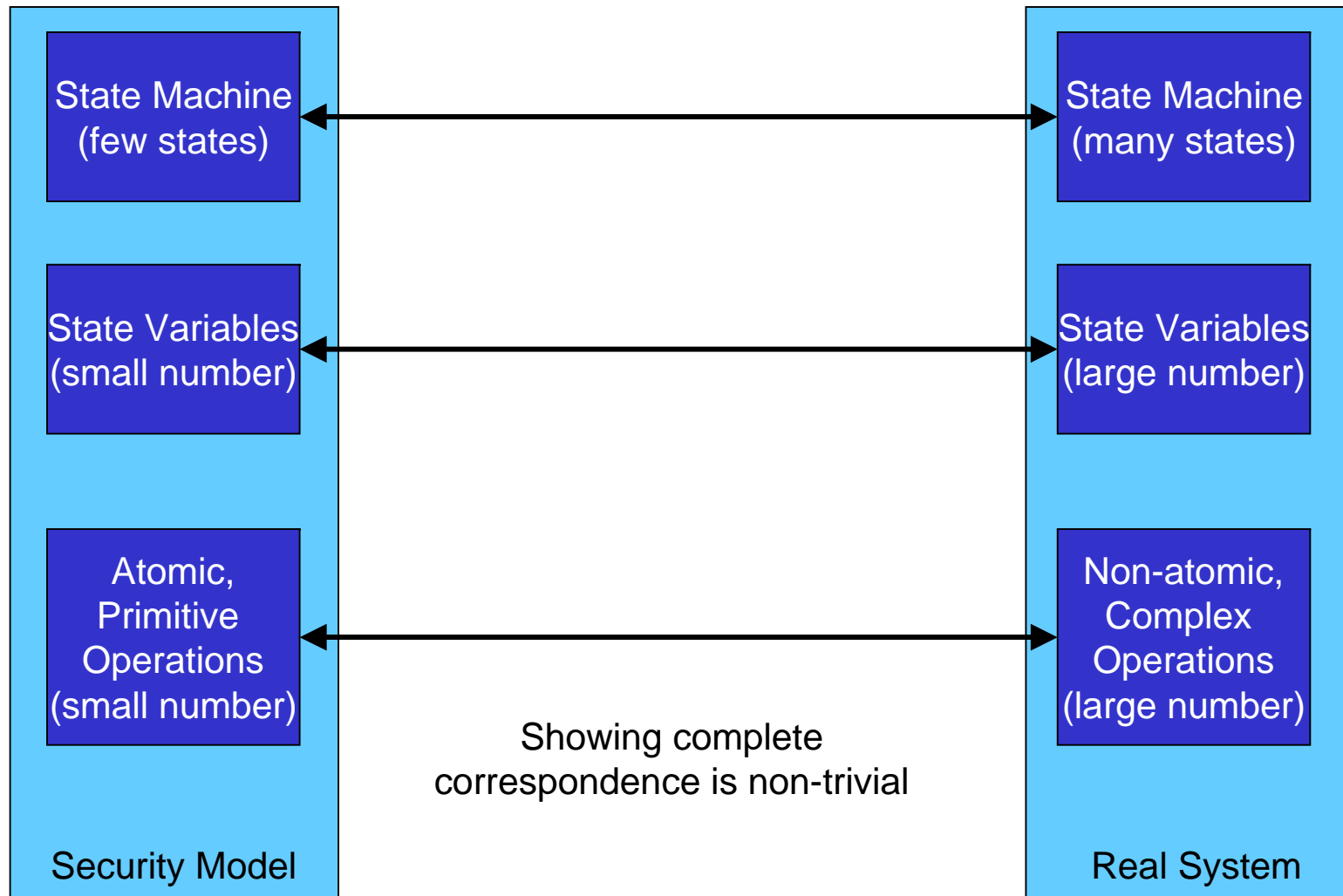
Function 6: *Append _data*(o, d)

if $o \in O$ and $w \in A(subj, to)$

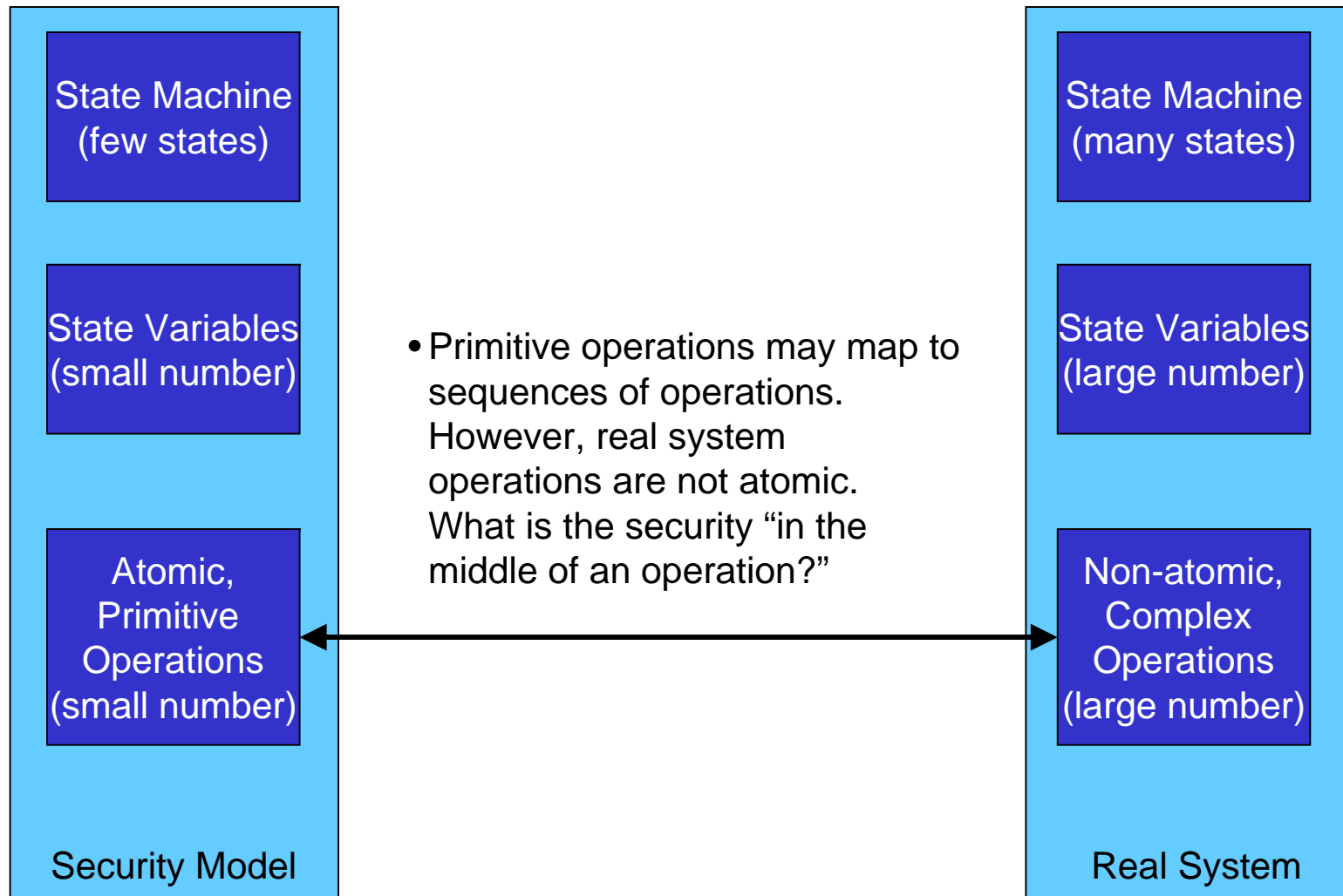
then $contents'(o) = contents(o) \cup \{d\}$

There might be a valid reason for allowing append to a file we can't, and never, read (e.g., an audit file)

Informal Model to System Correspondence



Informal Model to System Correspondence



Informal Model to System Correspondence

