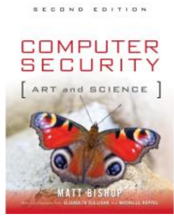
Foundational Results

Chapter 3

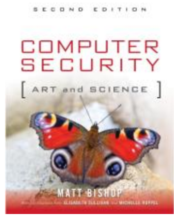


Overview

- Safety Question
- HRU Model
- Take-Grant Protection Model
- SPM, ESPM
    - Multiparent joint creation
- Expressive power
- Typed Access Matrix Model
- Comparing properties of models

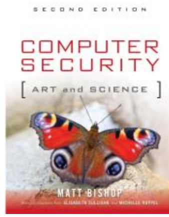
What Is "Secure"?

- Adding a generic right *r* where there was not one is "leaking"
  - In what follows, a right leaks if it was not present *initially*
  - Alternately: not present *in the previous state* (not discussed here)
- If a system *S*, beginning in initial state s_0, cannot leak right *r*, it is *safe with respect to the right r*
  - Otherwise it is called *unsafe with respect to the right r*



Safety Question

- Is there 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



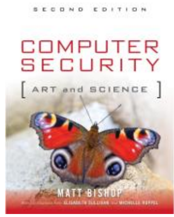
Mono-Operational Commands

- 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

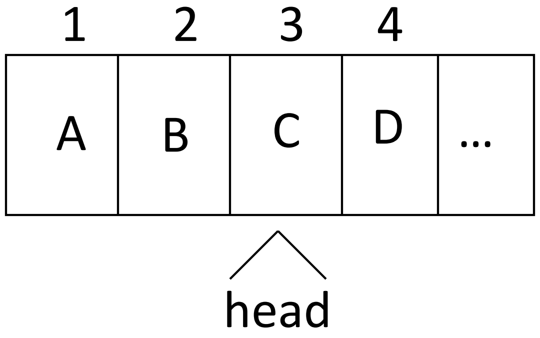
- Answer: *no*
- Sketch of proof:

  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

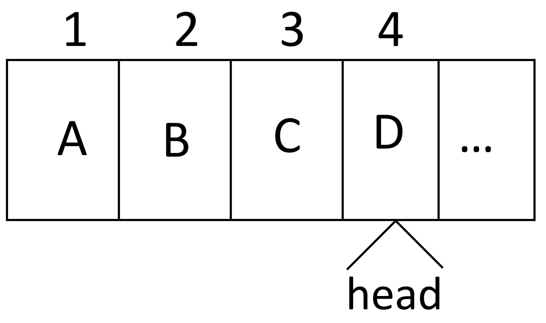
Mapping



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 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
```

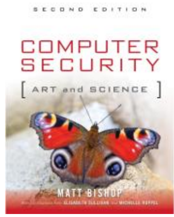
Mapping

1	2	3	4	
A	B	C	D	…

head

	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 $\delta(k_1, \text{D}) = (k_2, \text{Y}, \text{R})$ 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** $A[s_4, s_4]$;
  delete k_1 **from** $A[s_4, s_4]$;
  delete D **from** $A[s_4, s_4]$;
  enter Y **into** $A[s_4, s_4]$;
  create subject s_5;
  enter *own* **into** $A[s_4, s_5]$;
  enter *end* **into** $A[s_5, s_5]$;
  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 biconditional protection systems is decidable
- 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, …)

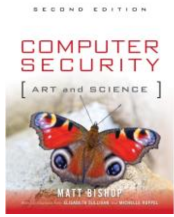
● subjects (users, processes, …)

⊗ 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'

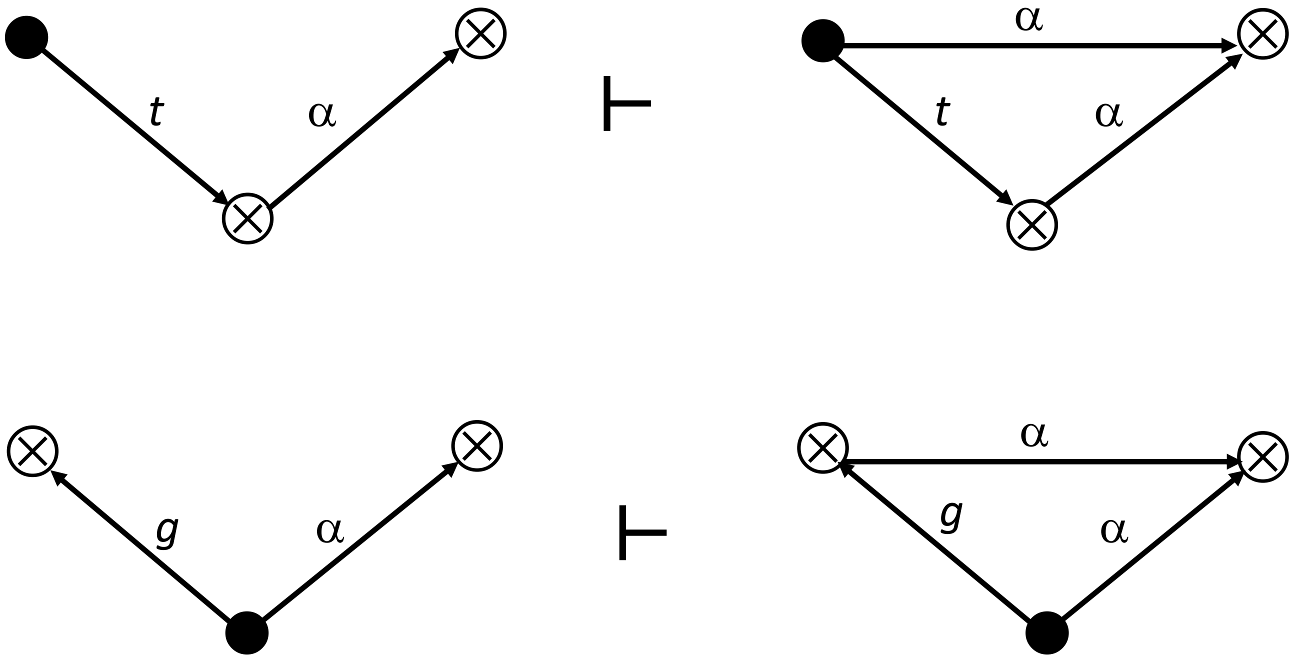
$R = \{ t, g, r, w, \ldots \}$ set of rights



Rules

take

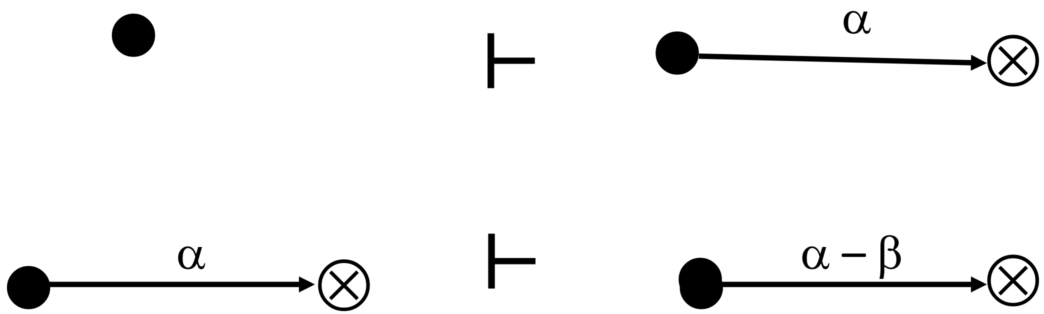
grant



More Rules

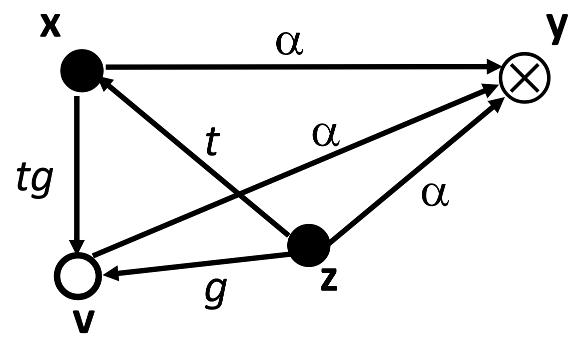
create

remove

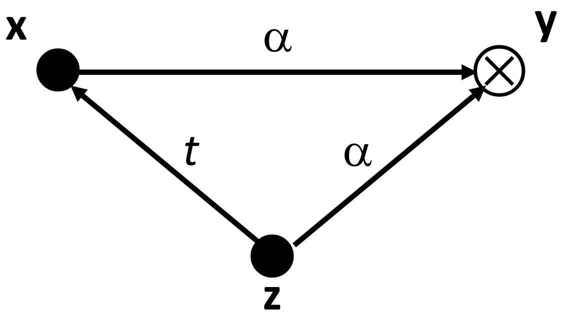


These four rules are called the *de jure* rules

Symmetry



$\vdash$

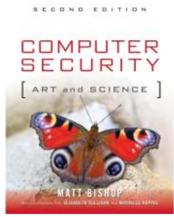


1. ***x*** creates (*tg* to new) ***v***
2. ***z*** takes (*g* to ***v***) from ***x***
3. ***z*** grants (α to ***y***) to ***v***
4. ***x*** takes (α to ***y***) from ***v***

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

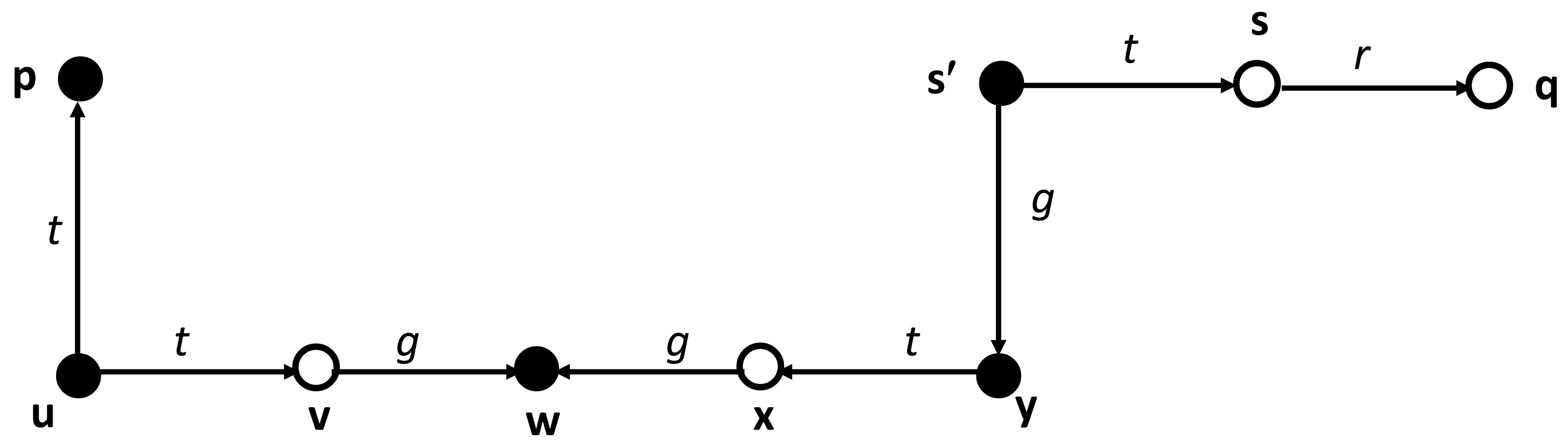
Bridges

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

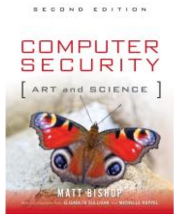
$$\{ \vec{t}^{*}, \vec{t}^{*}, \vec{t}^{*}\overleftarrow{g}\,\overleftarrow{t}^{*}, \vec{t}^{*}\vec{g}\,\overleftarrow{t}^{*} \}$$

  - rights can be transferred between the two endpoints
  - *not* an island as intermediate vertices are objects

Example



- islands { **p**, **u** } { **w** } { **y**, **s′** }
- bridges **uvw**; wxy
- initial span **p** (associated word ν)
- terminal span **s′s** (associated word $\vec{t}$)



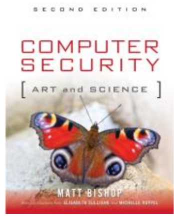
can•share Predicate

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.

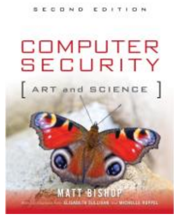
can•share Theorem

- *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



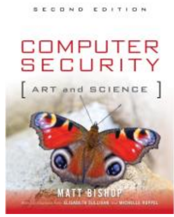
Outline of Proof

- **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



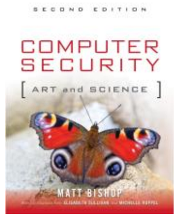
Example Interpretation

- ACM is generic
  - Can be applied in any situation
- Take-Grant has specific rules, rights
  - Can be applied in situations matching rules, rights
- Question: what states can evolve from a system that is modeled using the Take-Grant Model?



Take-Grant Generated Systems

- Theorem: G_0 protection graph with 1 vertex, no edges; R set of rights. Then $G_0 \vdash^* G$ iff:
    - G finite directed graph consisting of subjects, objects, edges
    - Edges labeled from nonempty subsets of R
    - At least one vertex in G has no incoming edges



Outline of Proof

$\Rightarrow$: By construction; G final graph in theorem

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

- Now 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 rights over $\mathbf{x}_j$, do "$\mathbf{x}_1$ grants (α to $\mathbf{x}_j$) to $\mathbf{x}_i$"
3. Let β be rights $\mathbf{x}_i$ has over $\mathbf{x}_j$ in G. Do "$\mathbf{x}_1$ removes (($\alpha \cup \{ g \} - \beta$ to) $\mathbf{x}_j$"

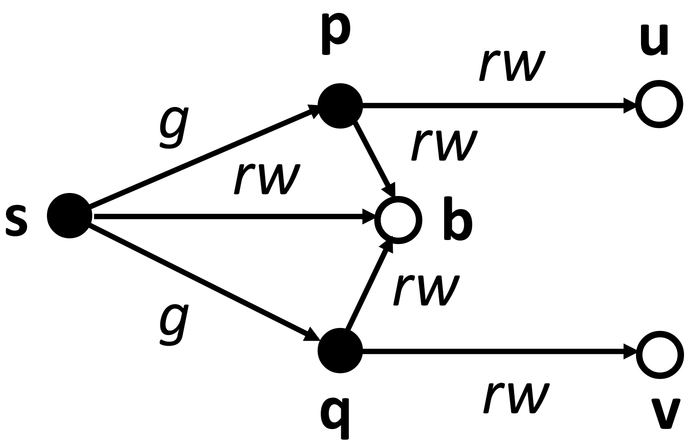
- Now G' is desired G

Outline of Proof

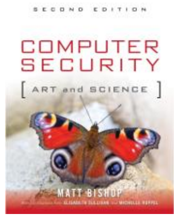
$\Leftarrow$: Let **v** be initial subject, and $G_0 \vdash^* G$

- Inspection of rules gives:
  - *G* is finite
  - *G* is a directed graph
  - Subjects and objects only
  - All edges labeled with nonempty subsets of *R*
- Limits of rules:
  - None allow vertices to be deleted so **v** in *G*
  - None add incoming edges to vertices without 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**
    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**

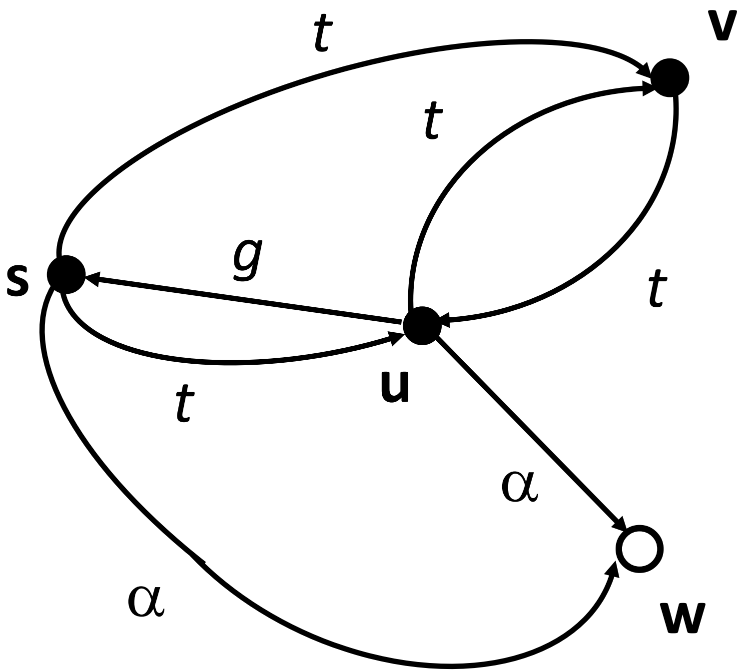


can•steal Predicate

Definition:

- *can•steal*(r, **x**, **y**, G_0) if, and only if, there is no edge from **x** to **y** labeled r in G_0, and the following hold simultaneously:
  - There is edge from **x** to **y** labeled r in G_n
  - There is a sequence of rule applications $\rho_1, \ldots, \rho_n$ such that $G_{i-1} \vdash G_i$ using ρ_i
  - 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

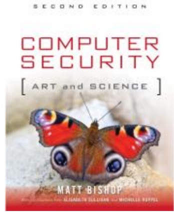


can•steal(α, **s**, **w**, G_0):

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

can•steal Theorem

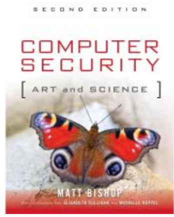
- *can•steal*(r, **x**, **y**, G_0) if, and only if, the following hold simultaneously:
  a) There is no edge from **x** to **y** labeled r in G_0
  b) There exists a subject **x**′ such that **x**′ = **x** or **x**′ initially spans to **x**
  c) There exists a vertex **s** with an edge labeled α to **y** in G_0
  d) *can•share*(t, **x**′, **s**, G_0) holds



Outline of Proof

$\Rightarrow$: Assume conditions hold

- **x** subject
  - **x** gets *t* rights to **s**, then takes α to **y** from **s**
- **x** object
  - *can•share*(*t*, **x′**, **s**, G_0) holds
  - If **x′** has no α edge to **y** in G_0, **x′** takes (α to **y**) from **s** and grants it to **x**
  - If **x′** has a edge to **y** in G_0, **x′** creates surrogate **x″**, gives it (*t* to **s**) and (*g* to **x″**); then **x″** takes (α to **y**) and grants it to **x**



Outline of Proof

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

- First two conditions immediate from definition of *can•steal*, *can•share*
- Third condition immediate from theorem of conditions for *can•share*
- Fourth condition: ρ minimal length sequence of rule applications deriving G_n from G_0; i smallest index such that $G_{i-1} \vdash G_i$ by rule ρ_i and adding α from some **p** to **y** in G_i
    - What is ρ_i?

Outline of Proof

- Not remove or create rule
  - **y** exists already
- Not grant rule
  - G_i first graph in which edge labeled α to **y** is added, so by definition of *can•share*, cannot be grant
- take rule: so *can•share*(t, **p**, **s**, G_0) holds
  - So is subject **s′** such that **s′** = **s** or terminally spans to **s**
  - Sequence of islands with **x′** $\in I_1$ and **s′** $\in I_n$
- Derive witness to *can•share*(t, **x′**, **s**, G_0) that does not use "**s** grants (α to **y**) to" anyone

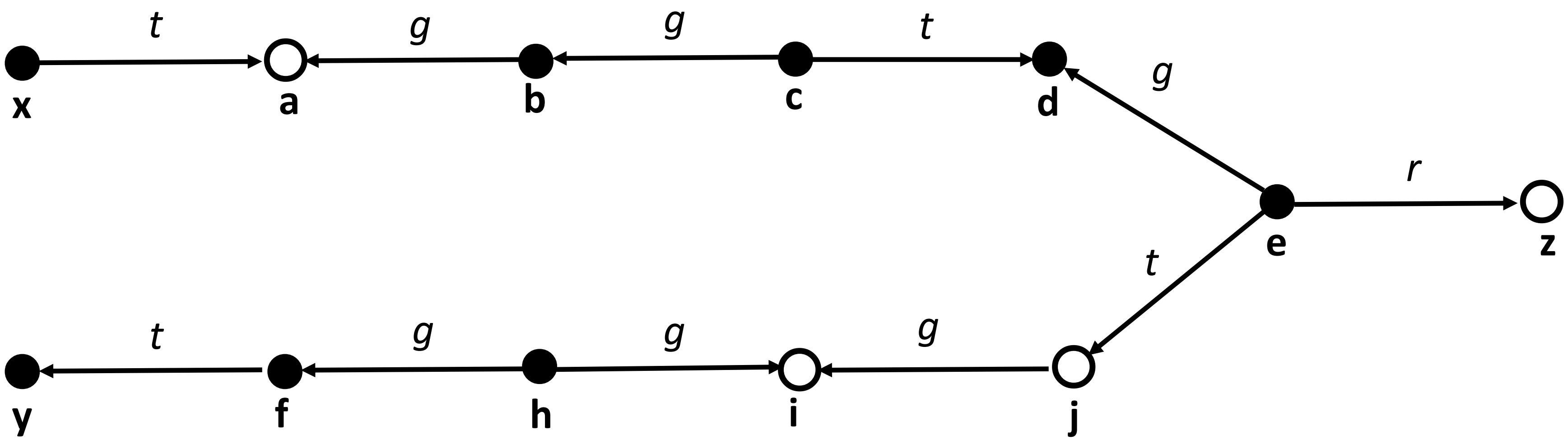
Conspiracy

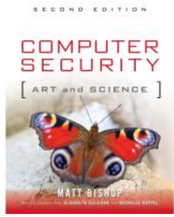
- Minimum number of actors to generate a witness for

  $$\textit{can}\bullet\textit{share}(\alpha, \mathbf{x}, \mathbf{y}, G_0)$$

- Access set describes the “reach” of a subject
- Deletion set is set of vertices that cannot be involved in a transfer of rights
- Build *conspiracy graph* to capture how rights flow, and derive actors from it

Example

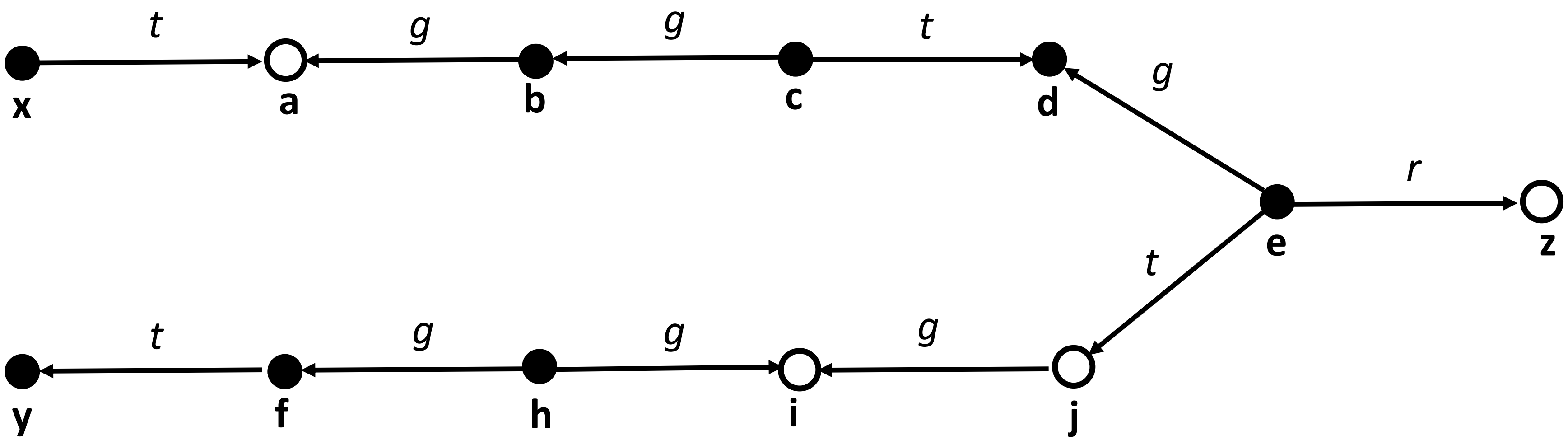




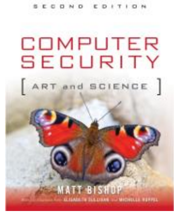
Access Set

- *Access set A(**y**) with focus* **y**: set of vertices:
  - { **y** }
  - { **x** | **y** initially spans to **x** }
  - { **x′** | **y** terminally spans to **x** }
- Idea is that focus can give rights to, or acquire rights from, a vertex in this set

Example



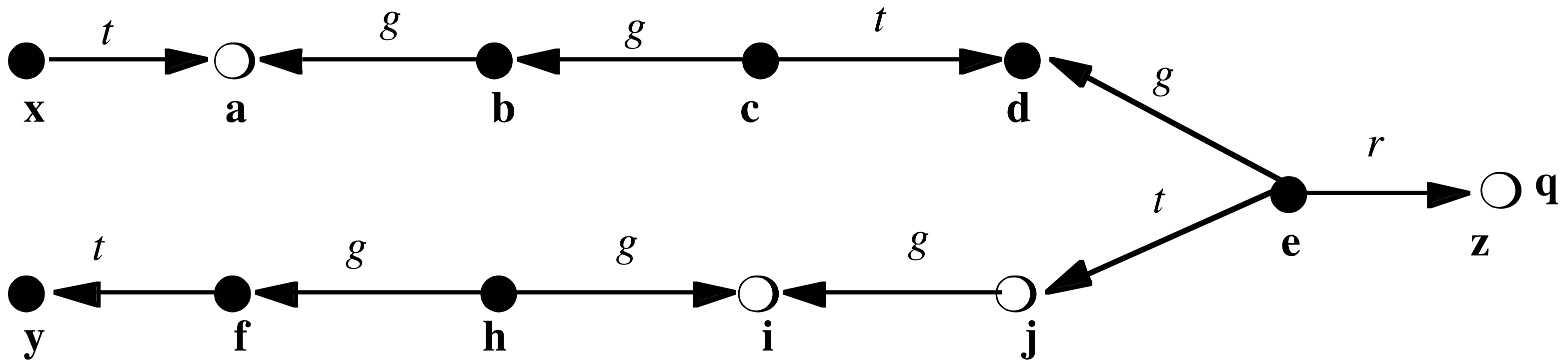
- A(**x**) = { **x**, **a** }
- A(**b**) = { **b**, **a** }
- A(**c**) = { **c**, **b**, **d** }
- A(**d**) = { **d** }
- A(**e**) = { **e**, **d**, **i**, **j** }
- A(**y**) = { **y** }
- A(**f**) = { **f**, **y** }
- A(**h**) = { **h**, **f**, **i** }



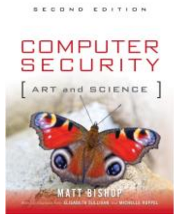
Deletion Set

- Deletion set $\delta(\mathbf{y}, \mathbf{y}')$: contains those vertices in $A(\mathbf{y}) \cap A(\mathbf{y}')$ such that:
  - **y** initially spans to **z** and **y′** terminally spans to **z**;
  - **y** terminally spans to **z** and **y′** initially spans to **z**;
  - $\mathbf{z} = \mathbf{y}$
  - $\mathbf{z} = \mathbf{y}'$
- Idea is that rights can be transferred between **y** and **y′** if this set non-empty

Example



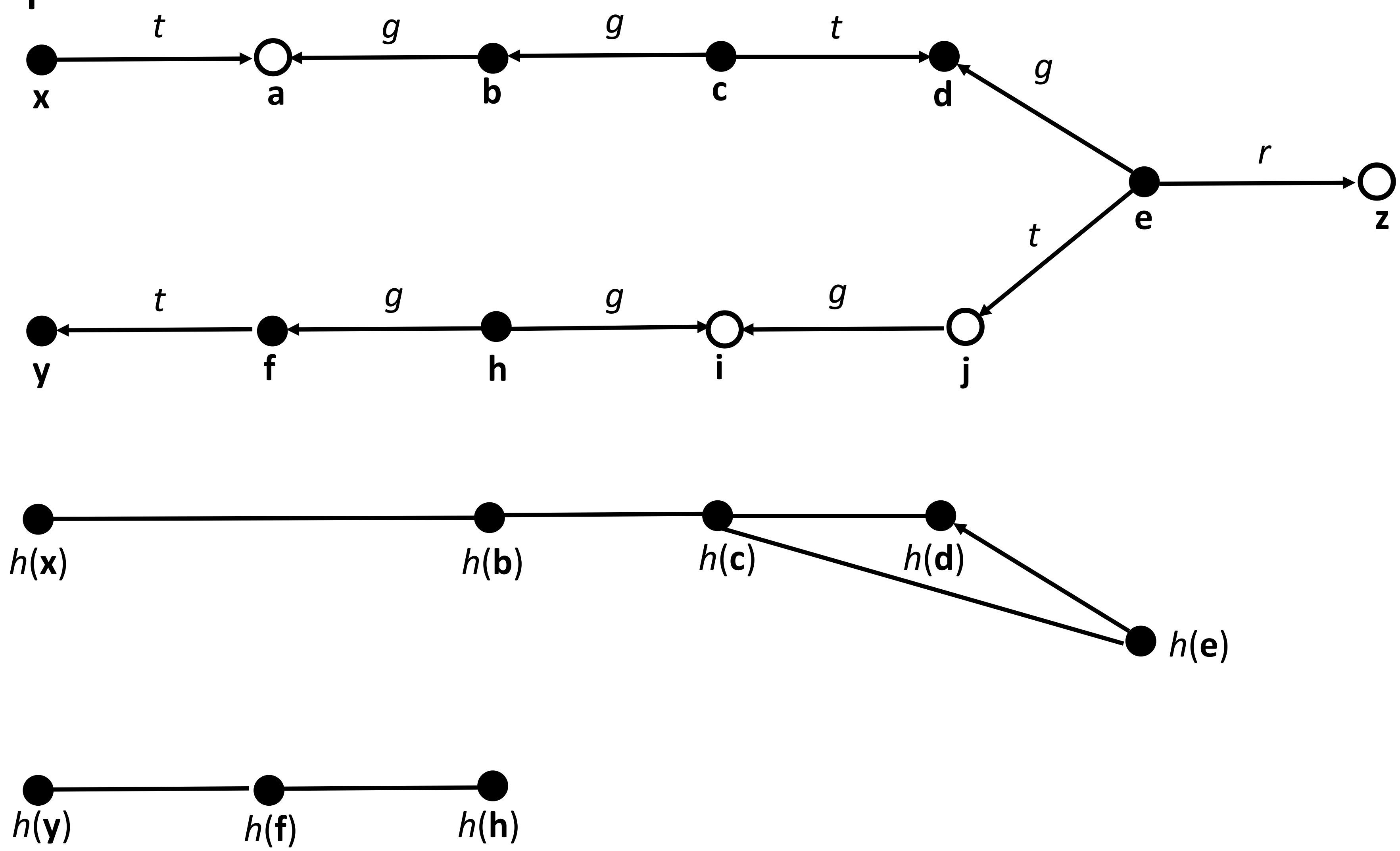
- δ(**x**, **b**) = { **a** }
- δ(**b**, **c**) = { **b** }
- δ(**c**, **d**) = { **d** }
- δ(**c**, **e**) = { **d** }
- δ(**d**, **e**) = { **d** }
- δ(**y**, **f**) = { **y** }
- δ(**h**, **f**) = { **f** }

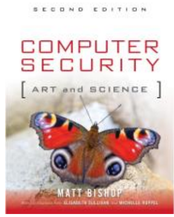


Conspiracy Graph

- Abstracted graph H from G_0:
  - Each subject $\mathbf{x} \in G_0$ corresponds to a vertex $h(\mathbf{x}) \in H$
  - If $\delta(\mathbf{x}, \mathbf{y}) \neq \varnothing$, there is an edge between $h(\mathbf{x})$ and $h(\mathbf{y})$ in H
- Idea is that if $h(\mathbf{x})$, $h(\mathbf{y})$ are connected in H, then rights can be transferred between **x** and **y** in G_0

Example

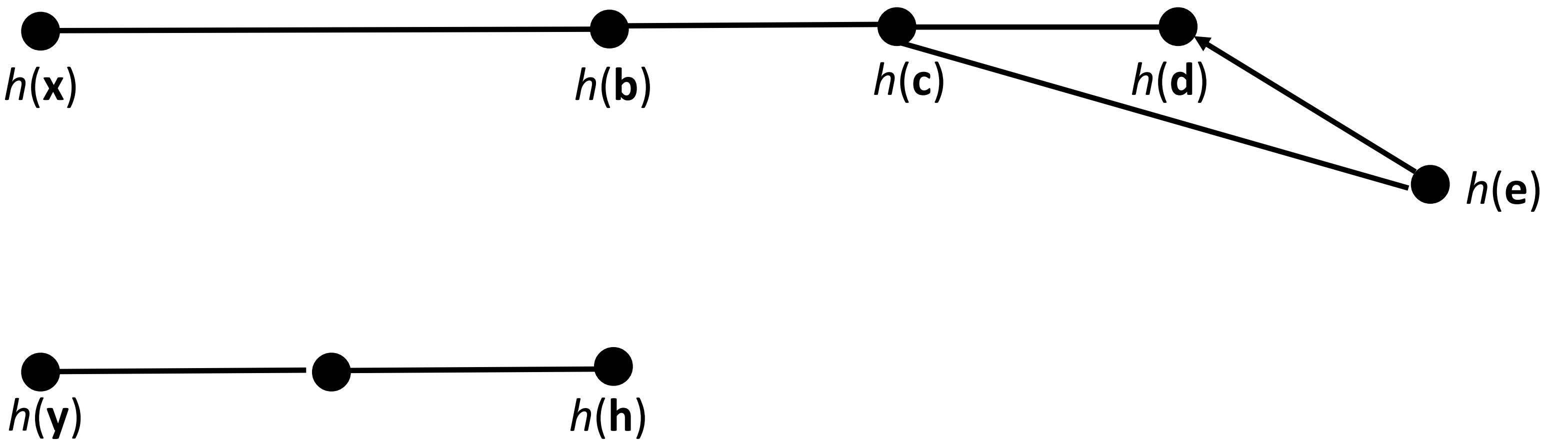




Results

- $I(\mathbf{x})$: $h(\mathbf{x})$, all vertices $h(\mathbf{y})$ such that $\mathbf{y}$ initially spans to $\mathbf{x}$
- $T(\mathbf{x})$: $h(\mathbf{x})$, all vertices $h(\mathbf{y})$ such that $\mathbf{y}$ terminally spans to $\mathbf{x}$
- Theorem: *can•share*$(\alpha, \mathbf{x}, \mathbf{y}, G_0)$ iff there exists a path from some $h(\mathbf{p})$ in $I(\mathbf{x})$ to some $h(\mathbf{q})$ in $T(\mathbf{y})$
- Theorem: l vertices on shortest path between $h(\mathbf{p})$, $h(\mathbf{q})$ in above theorem; l conspirators necessary and sufficient to witness

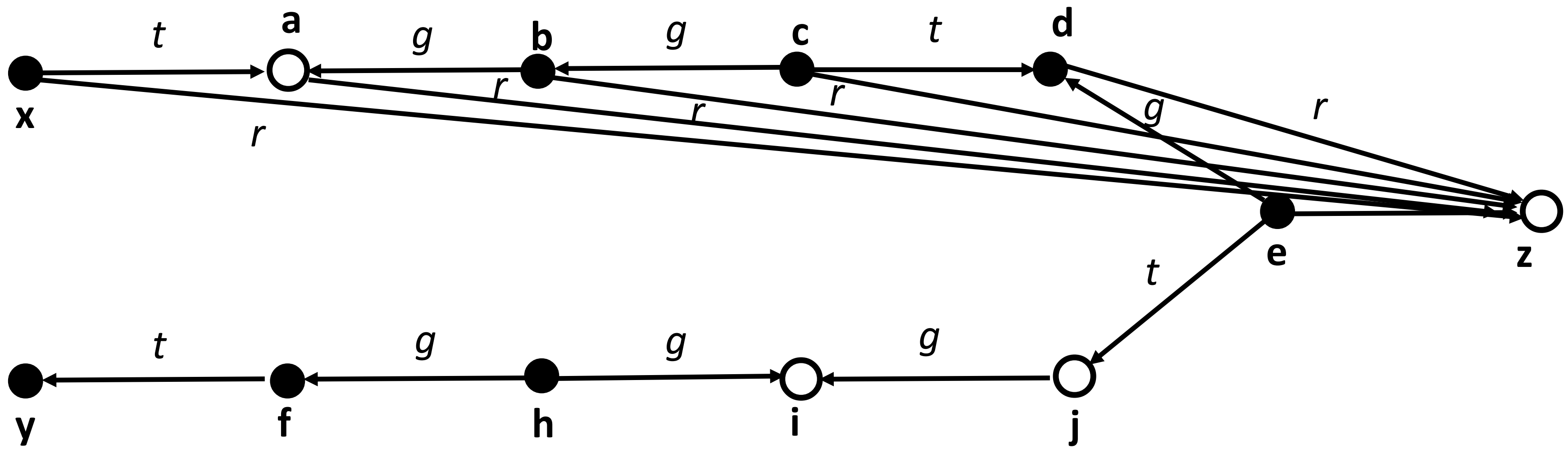
Example: Conspirators



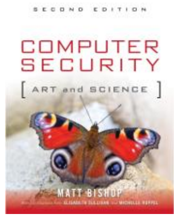
- $I(\mathbf{x}) = \{ h(\mathbf{x}) \}$, $T(\mathbf{z}) = \{ h(\mathbf{e}) \}$
- Path between $h(\mathbf{x})$, $h(\mathbf{e})$ so *can•share*$(r, \mathbf{x}, \mathbf{z}, G_0)$
- Shortest path between $h(\mathbf{x})$, $h(\mathbf{e})$ has 4 vertices

$\Rightarrow$ Conspirators are **e**, **c**, **b**, **x**

Example: Witness



1. **e** grants (*r* to **z**) to **d**
2. **c** takes (*r* to **z**) from **d**
3. **c** grants (*r* to **z**) to **b**
4. **b** grants (*r* to **z**) to **a**
5. **x** takes (*r* to **z**) from **a**

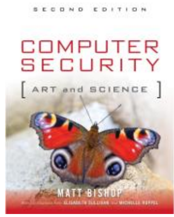


Key Question

- Characterize class of models for which safety is decidable
  - Existence: Take-Grant Protection Model is a member of such a class
  - Universality: In general, question undecidable, so for some models it is not decidable
- What is the dividing line?

Schematic Protection Model

- Type-based model
  - Protection type: entity label determining how control rights affect the entity
    - Set at creation and cannot be changed
  - Ticket: description of a single right over an entity
    - Entity has sets of tickets (called a *domain*)
    - Ticket is **X**/*r*, where **X** is entity and *r* right
  - Functions determine rights transfer
    - Link: are source, target “connected”?
    - Filter: is transfer of ticket authorized?



Link Predicate

- Idea: $link_i$(**X**, **Y**) if **X** can assert some control right over **Y**
- Conjunction of disjunction of:
  - $\mathbf{X}/z \in dom(\mathbf{X})$
  - $\mathbf{X}/z \in dom(\mathbf{Y})$
  - $\mathbf{Y}/z \in dom(\mathbf{X})$
  - $\mathbf{Y}/z \in dom(\mathbf{Y})$
  - **true**

Examples

- Take-Grant:

  $\mathit{link}(\mathbf{X}, \mathbf{Y}) = \mathbf{Y}/g \in \mathit{dom}(\mathbf{X}) \vee \mathbf{X}/t \in \mathit{dom}(\mathbf{Y})$

- Broadcast:

  $\mathit{link}(\mathbf{X}, \mathbf{Y}) = \mathbf{X}/b \in \mathit{dom}(\mathbf{X})$

- Pull:

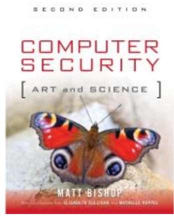
  $\mathit{link}(\mathbf{X}, \mathbf{Y}) = \mathbf{Y}/p \in \mathit{dom}(\mathbf{Y})$

Filter Function

- Range is set of copyable tickets
  - Entity type, right
- Domain is subject pairs
- Copy a ticket $\mathbf{X}/r{:}c$ from $dom(\mathbf{Y})$ to $dom(\mathbf{Z})$
  - $\mathbf{X}/rc \in dom(\mathbf{Y})$
  - $link_i(\mathbf{Y}, \mathbf{Z})$
  - $\tau(\mathbf{Y})/r{:}c \in f_i(\tau(\mathbf{Y}), \tau(\mathbf{Z}))$
- One filter function per link function

Example

- $f(\tau(\mathbf{Y}), \tau(\mathbf{Z})) = T \times R$
  - Any ticket can be transferred (if other conditions met)
- $f(\tau(\mathbf{Y}), \tau(\mathbf{Z})) = T \times RI$
  - Only tickets with inert rights can be transferred (if other conditions met)
- $f(\tau(\mathbf{Y}), \tau(\mathbf{Z})) = \varnothing$
  - No tickets can be transferred

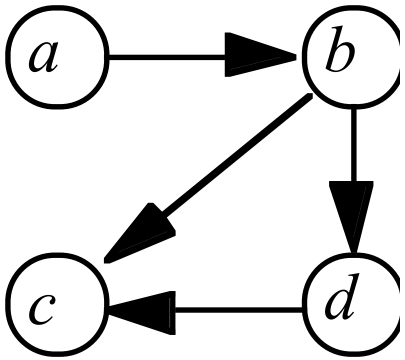


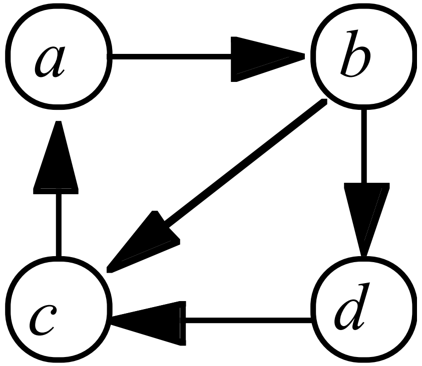
Example

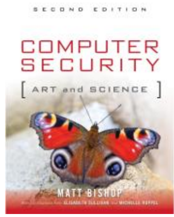
- Take-Grant Protection Model
  - TS = { subjects }, TO = { objects }
  - $RC = \{ tc, gc \}$, $RI = \{ rc, wc \}$
  - $link(\mathbf{p}, \mathbf{q}) = \mathbf{p}/t \in dom(\mathbf{q}) \vee \mathbf{q}/g \in dom(\mathbf{p})$
  - $f(subject, subject) = \{ subject, object \} \times \{ tc, gc, rc, wc \}$

Create Operation

- Must handle type, tickets of new entity
- Relation *cc*(*a*, *b*) [*cc* for *can-create*]
    - Subject of type *a* can create entity of type *b*
- Rule of acyclic creates:

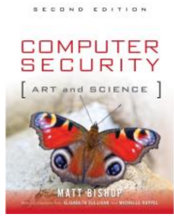






Types

- *cr*(*a*, *b*): tickets created when subject of type *a* creates entity of type *b* [*cr* for *create-rule*]
- **B** object: $cr(a, b) \subseteq \{ b/r{:}c \in RI \}$
  - **A** gets **B**/$r{:}c$ iff $b/r{:}c \in cr(a, b)$
- **B** subject: *cr*(*a*, *b*) has two subsets
  - $cr_P(a, b)$ added to **A**, $cr_C(a, b)$ added to **B**
  - **A** gets **B**/$r{:}c$ if $b/r{:}c \in cr_P(a, b)$
  - **B** gets **A**/$r{:}c$ if $a/r{:}c \in cr_C(a, b)$

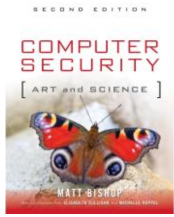


Non-Distinct Types

$cr(a, a)$: who gets what?

- $self/r{:}c$ are tickets for creator
- $a/r{:}c$ tickets for created

$cr(a, a) = \{\, a/r{:}c,\ self/r{:}c \mid r{:}c \in R\}$



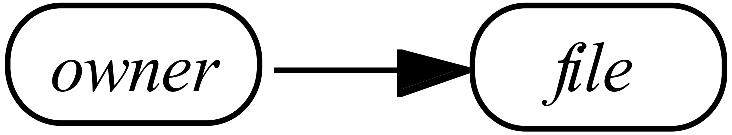
Attenuating Create Rule

$cr(a, b)$ attenuating if:

1. $cr_C(a, b) \subseteq cr_P(a, b)$ and
2. $a/r{:}c \in cr_P(a, b) \Rightarrow self/r{:}c \in cr_P(a, b)$

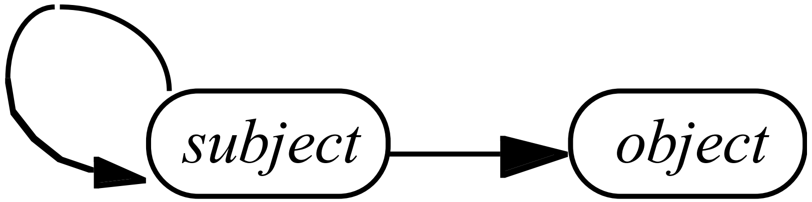
Example: Owner-Based Policy

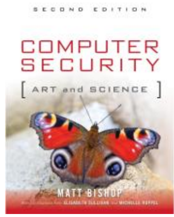
- Users can create files, creator can give itself any inert rights over file
  - $cc = \{ \ (\ user\ ,\ \ file\ )\ \}$
  - $cr(user, file) = \{ \ file/r{:}c \mid r \in RI \ \}$
- Attenuating, as graph is acyclic, loop free



Example: Take-Grant

- Say subjects create subjects (type *s*), objects (type *o*), but get only inert rights over latter
  - $cc = \{ (s, s), (s, o) \}$
  - $cr_C(a, b) = \varnothing$
  - $cr_P(s, s) = \{s/tc, s/gc, s/rc, s/wc\}$
  - $cr_P(s, o) = \{s/rc, s/wc\}$
- Not attenuating, as no *self* tickets provided; *subject* creates *subject*



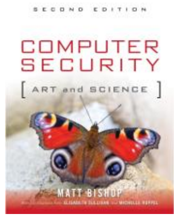


Safety Analysis

- Goal: identify types of policies with tractable safety analyses
- Approach: derive a state in which additional entries, rights do not affect the analysis; then analyze this state
    - Called a *maximal state*

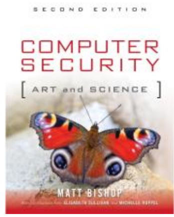
Definitions

- System begins at initial state
- Authorized operation causes *legal transition*
- Sequence of legal transitions moves system into final state
  - This sequence is a *history*
  - Final state is *derivable* from history, initial state



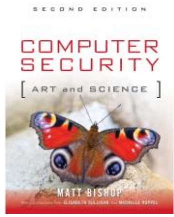
More Definitions

- States represented by ^{h}
- Set of subjects SUB^h, entities ENT^h
- Link relation in context of state h is $link^h$
- Dom relation in context of state h is dom^h



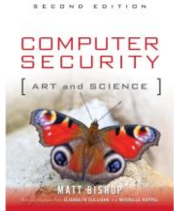
$path^h$(X,Y)

- **X**, **Y** connected by one link or a sequence of links
- Formally, either of these hold:
  - for some i, $link_i^h(\mathbf{X}, \mathbf{Y})$; or
  - there is a sequence of subjects $\mathbf{X}_0, \ldots, \mathbf{X}_n$ such that $link_i^h(\mathbf{X}, \mathbf{X}_0)$, $link_i^h(\mathbf{X}_n, \mathbf{Y})$, and for $k = 1, \ldots, n$, $link_i^h(\mathbf{X}_{k-1}, \mathbf{X}_k)$
- If multiple such paths, refer to $path_j^h(\mathbf{X}, \mathbf{Y})$



Capacity *cap*(*path*^{h}(**X**,**Y**))

- Set of tickets that can flow over *path*^{h}(**X**,**Y**)
  - If $link_i^h$(**X**,**Y**): set of tickets that can be copied over the link (i.e., $f_i(\tau(\mathbf{X}), \tau(\mathbf{Y}))$)
  - Otherwise, set of tickets that can be copied over *all* links in the sequence of links making up the *path*^{h}(**X**,**Y**)
- Note: all tickets (except those for the final link) *must* be copyable



Flow Function

- Idea: capture flow of tickets around a given state of the system
- Let there be *m* $path^h$s between subjects **X** and **Y** in state *h*. Then *flow function*

$$flow^h: SUB^h \times SUB^h \rightarrow 2^{T \times R}$$

is:

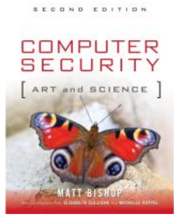
$$flow^h(\mathbf{X},\mathbf{Y}) = \cup_{i=1,\ldots,m}\ cap(path_i^{\,h}(\mathbf{X},\mathbf{Y}))$$

Properties of Maximal State

- Maximizes flow between all pairs of subjects
  - State is called *
  - Ticket in *flow**(**X**,**Y**) means there exists a sequence of operations that can copy the ticket from **X** to **Y**
- Questions
  - Is maximal state unique?
  - Does every system have one?

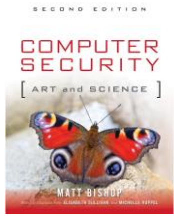
Formal Definition

- Definition: $g \leq_0 h$ holds iff for all **X**, **Y** $\in SUB^0$, $flow^g(\mathbf{X},\mathbf{Y}) \subseteq flow^h(\mathbf{X},\mathbf{Y})$.
  - Note: if $g \leq_0 h$ and $h \leq_0 g$, then g, h equivalent
  - Defines set of equivalence classes on set of derivable states
- Definition: for a given system, state m is maximal iff $h \leq_0 m$ for every derivable state h
- Intuition: flow function contains all tickets that can be transferred from one subject to another
  - All maximal states in same equivalence class



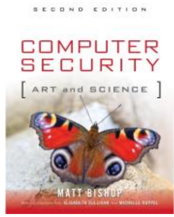
Maximal States

- Lemma. Given arbitrary finite set of states H, there exists a derivable state m such that for all $h \in H$, $h \leq_0 m$
- Outline of proof: induction
  - Basis: $H = \varnothing$; trivially true
  - Step: $|H'| = n + 1$, where $H' = G \cup \{h\}$. By IH, there is a $g \in G$ such that $x \leq_0 g$ for all $x \in G$.



Outline of Proof

- M interleaving histories of g, h which:
  - Preserves relative order of transitions in g, h
  - Omits second create operation if duplicated
- M ends up at state m
- If $path^g(\mathbf{X},\mathbf{Y})$ for $\mathbf{X}$, $\mathbf{Y} \in SUB^g$, $path^m(\mathbf{X},\mathbf{Y})$
  - So $g \leq_0 m$
- If $path^h(\mathbf{X},\mathbf{Y})$ for $\mathbf{X}$, $\mathbf{Y} \in SUB^h$, $path^m(\mathbf{X},\mathbf{Y})$
  - So $h \leq_0 m$
- Hence m maximal state in H'



Answer to Second Question

- Theorem: every system has a maximal state *
- Outline of proof: *K* is set of derivable states containing exactly one state from each equivalence class of derivable states
  - Consider **X**, **Y** in SUB^0. Flow function's range is $2^{T \times R}$, so can take at most $2^{|T \times R|}$ values. As there are $|SUB^0|^2$ pairs of subjects in SUB^0, at most $2^{|T \times R|} |SUB^0|^2$ distinct equivalence classes; so *K* is finite
- Result follows from lemma

Safety Question

- In this model:

  Is it possible to have a derivable state with **X**/*r*:*c* in *dom*(**A**), or does there exist a subject **B** with ticket **X**/*rc* in the initial state or which can demand **X**/*rc* and τ(**X**)/*r*:*c* in *flow**(**B**,**A**)?

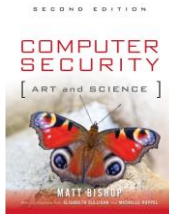
- To answer: construct maximal state and test
  - Consider acyclic attenuating schemes; how do we construct maximal state?

Intuition

- Consider state *h*.
- State *u* corresponds to *h* but with minimal number of new entities created such that maximal state *m* can be derived with no create operations
  - So if in history from *h* to *m*, subject **X** creates two entities of type *a*, in *u* only one would be created; surrogate for both
- *m* can be derived from *u* in polynomial time, so if *u* can be created by adding a finite number of subjects to *h*, safety question decidable.

Fully Unfolded State

- State *u* derived from state 0 as follows:
  - delete all loops in *cc*; new relation *cc*′
  - mark all subjects as folded
  - while any **X** $\in SUB^0$ is folded
    - mark it unfolded
    - if **X** can create entity **Y** of type *y*, it does so (call this the *y*-surrogate of **X**); if entity **Y** $\in SUB^g$, mark it folded
  - if any subject in state *h* can create an entity of its own type, do so
- Now in state *u*



Termination

- First loop terminates as SUB^0 finite
- Second loop terminates:
  - Each subject in SUB^0 can create at most | TS | children, and | TS | is finite
  - Each folded subject in | SUB^i | can create at most | TS | – i children
  - When i = | TS |, subject cannot create more children; thus, folded is finite
  - Each loop removes one element
- Third loop terminates as SUB^h is finite

Surrogate

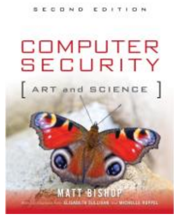
- Intuition: surrogate collapses multiple subjects of same type into single subject that acts for all of them
- Definition: given initial state 0, for every derivable state h define *surrogate function* $\sigma{:}ENT^h \rightarrow ENT^h$ by:
  - if **X** in ENT^0, then σ(**X**) = **X**
  - if **Y** creates **X** and τ(**Y**) = τ(**X**), then σ(**X**) = σ(**Y**)
  - if **Y** creates **X** and τ(**Y**) ≠ τ(**X**), then σ(**X**) = τ(**Y**)-surrogate of σ(**Y**)

Implications

- $\tau(\sigma(\mathbf{X})) = \tau(\mathbf{X})$
- If $\tau(\mathbf{X}) = \tau(\mathbf{Y})$, then $\sigma(\mathbf{X}) = \sigma(\mathbf{Y})$
- If $\tau(\mathbf{X}) \neq \tau(\mathbf{Y})$, then
  - $\sigma(\mathbf{X})$ creates $\sigma(\mathbf{Y})$ in the construction of u
  - $\sigma(\mathbf{X})$ creates entities $\mathbf{X}'$ of type $\tau(\mathbf{X}') = \tau(\sigma(\mathbf{X}))$
- From these, for a system with an acyclic attenuating scheme, if **X** creates **Y**, then tickets that would be introduced by pretending that $\sigma(\mathbf{X})$ creates $\sigma(\mathbf{Y})$ are in $dom^u(\sigma(\mathbf{X}))$ and $dom^u(\sigma(\mathbf{Y}))$

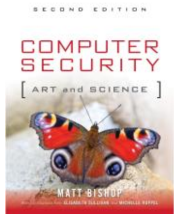
Deriving Maximal State

- Idea
  - Reorder operations so that all creates come first and replace history with equivalent one using surrogates
  - Show maximal state of new history is also that of original history
  - Show maximal state can be derived from initial state



Reordering

- *H* legal history deriving state *h* from state 0
- Order operations: first create, then demand, then copy operations
- Build new history *G* from *H* as follows:
  - Delete all creates
  - "**X** demands **Y**/*r*:*c*" becomes "σ(**X**) demands σ(**Y**)/*r*:*c*"
  - "**Y** copies **X** /*r*:*c* from **Y**" becomes "σ(**Y**) copies σ(**X**)/*r*:*c* from σ(**Y**)"

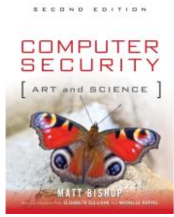


Tickets in Parallel

- Lemma
    - All transitions in G legal; if $\mathbf{X}/r{:}c \in dom^h(Y)$, then $\sigma(\mathbf{X})/r{:}c \in dom^h(\sigma(\mathbf{Y}))$
- Outline of proof: induct on number of copy operations in H

Basis

- H has create, demand only; so G has demand only. s preserves type, so by construction every demand operation in G legal.
- 3 ways for $\mathbf{X}/r{:}c$ to be in $dom^h(\mathbf{Y})$:
  - $\mathbf{X}/r{:}c \in dom^0(\mathbf{Y})$ means $\mathbf{X}, \mathbf{Y} \in ENT^0$, so trivially $\sigma(\mathbf{X})/r{:}c \in dom^g(\sigma(\mathbf{Y}))$ holds
  - A create added $\mathbf{X}/r{:}c \in dom^h(\mathbf{Y})$: previous lemma says $\sigma(\mathbf{X})/r{:}c \in dom^g(\sigma(\mathbf{Y}))$ holds
  - A demand added $\mathbf{X}/r{:}c \in dom^h(\mathbf{Y})$: corresponding demand operation in G gives $\sigma(\mathbf{X})/r{:}c \in dom^g(\sigma(\mathbf{Y}))$

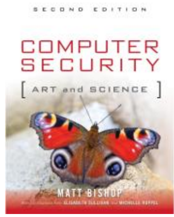


Hypothesis

- Claim holds for all histories with k copy operations
- History H has k+1 copy operations
  - H' initial sequence of H composed of k copy operations
  - h' state derived from H'

Step

- G' sequence of modified operations corresponding to H'; g' derived state
  - G' legal history by hypothesis
- Final operation is "Z copied X/r:c from Y"
  - So h, h' differ by at most $\mathbf{X}/r{:}c \in dom^h(\mathrm{Z})$
  - Construction of G means final operation is $\sigma(\mathbf{X})/r{:}c \in dom^g(\sigma(\mathbf{Y}))$
- Proves second part of claim



Step

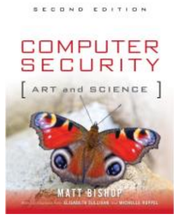
- H' legal, so for H to be legal, we have:
    1. $\mathbf{X}/rc \in dom^{h'}(\mathbf{Y})$
    2. $link_i^{h'}(\mathbf{Y}, \mathbf{Z})$
    3. $\tau(\mathbf{X}/r{:}c) \in f_i(\tau(\mathbf{Y}), \tau(\mathbf{Z}))$
- By IH, 1, 2, as $\mathbf{X}$/r:c $\in dom^{h'}(\mathbf{Y})$,

$$\sigma(\mathbf{X})/r{:}c \in dom^{g'}(\sigma(\mathbf{Y})) \text{ and } link_i^{g'}(\sigma(\mathbf{Y}), \sigma(\mathbf{Z}))$$

- As σ preserves type, IH and 3 imply

$$\tau(\sigma(\mathbf{X})/r{:}c) \in f_i(\tau((\sigma(\mathbf{Y})), \tau(\sigma(\mathbf{Z})))$$

- IH says G' legal, so G is legal



Corollary

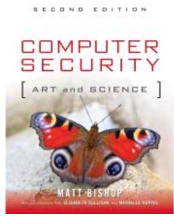
- If $link_i^h(\mathbf{X}, \mathbf{Y})$, then $link_i^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y}))$

Main Theorem

- System has acyclic attenuating scheme
- For every history *H* deriving state *h* from initial state, there is a history *G* without create operations that derives *g* from the fully unfolded state *u* such that

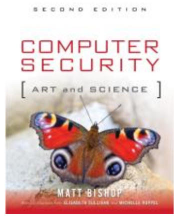
$$(\forall \mathbf{X},\mathbf{Y} \in SUB^h)[flow^h(\mathbf{X}, \mathbf{Y}) \subseteq flow^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y}))]$$

- Meaning: any history derived from an initial state can be simulated by corresponding history applied to the fully unfolded state derived from the initial state



Proof

- Outline of proof: show that every *path*^{h}(**X**,**Y**) has corresponding *path*^{g}(σ(**X**), σ(**Y**)) such that *cap*(*path*^{h}(**X**,**Y**)) = *cap*(*path*^{g}(σ(**X**), σ(**Y**)))
  - Then corresponding sets of tickets flow through systems derived from *H* and *G*
  - As initial states correspond, so do those systems
- Proof by induction on number of links

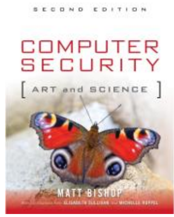


Basis and Hypothesis

- Length of $path^h(\mathbf{X}, \mathbf{Y}) = 1$. By definition of $path^h$, $link_i^h(\mathbf{X}, \mathbf{Y})$, hence $link_i^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y}))$. As σ preserves type, this means

$$cap(path^h(\mathbf{X}, \mathbf{Y})) = cap(path^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y})))$$

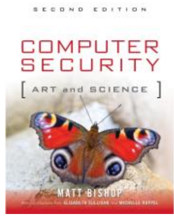
- Now assume this is true when $path^h(\mathbf{X}, \mathbf{Y})$ has length k



Step

- Let $path^h(\mathbf{X}, \mathbf{Y})$ have length k+1. Then there is a **Z** such that $path^h(\mathbf{X}, \mathbf{Z})$ has length k and $link_j^h(\mathbf{Z}, \mathbf{Y})$.
- By IH, there is a $path^g(\sigma(\mathbf{X}), \sigma(\mathbf{Z}))$ with same capacity as $path^h(\mathbf{X}, \mathbf{Z})$
- By corollary, $link_j^g(\sigma(\mathbf{Z}), \sigma(\mathbf{Y}))$
- As σ preserves type, there is $path^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y}))$ with

$$cap(path^h(\mathbf{X}, \mathbf{Y})) = cap(path^g(\sigma(\mathbf{X}), \sigma(\mathbf{Y})))$$



Implication

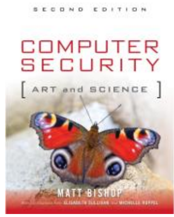
- Let maximal state corresponding to *v* be *#u*
  - Deriving history has no creates
  - By theorem,

$$(\forall \mathbf{X},\mathbf{Y} \in SUB^h)[flow^h(\mathbf{X}, \mathbf{Y}) \subseteq flow^{\#u}(\sigma(\mathbf{X}), \sigma(\mathbf{Y}))]$$

  - If $\mathbf{X} \in SUB^0$, $\sigma(\mathbf{X}) = \mathbf{X}$, so:

$$(\forall \mathbf{X},\mathbf{Y} \in SUB^0)[flow^h(\mathbf{X}, \mathbf{Y}) \subseteq flow^{\#u}(\mathbf{X}, \mathbf{Y})]$$

- So *#u* is maximal state for system with acyclic attenuating scheme
  - *#u* derivable from *u* in time polynomial to $|SUB^u|$
  - Worst case computation for $flow^{\#u}$ is exponential in $|TS|$

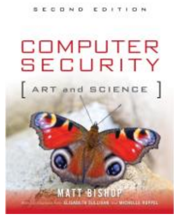


Safety Result

- If the scheme is acyclic and attenuating, the safety question is decidable

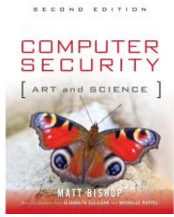
Expressive Power

- How do the sets of systems that models can describe compare?
  - If HRU equivalent to SPM, SPM provides more specific answer to safety question
  - If HRU describes more systems, SPM applies only to the systems it can describe



HRU *vs*. SPM

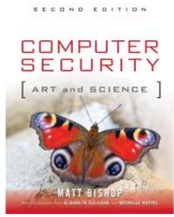
- SPM more abstract
  - Analyses focus on limits of model, not details of representation
- HRU allows revocation
  - SMP has no equivalent to delete, destroy
- HRU allows multiparent creates
  - SMP cannot express multiparent creates easily, and not at all if the parents are of different types because *can•create* allows for only one type of creator



Multiparent Create

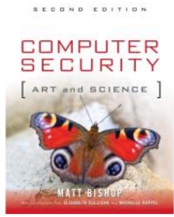
- Solves mutual suspicion problem
  - Create proxy jointly, each gives it needed rights
- In HRU:

command *multicreate*(s_0, s_1, o)
if r **in** $a[s_0, s1]$ **and** r **in** $a[s_1, s_0]$
then
  create object o;
  enter r **into** $a[s_0, o]$;
  enter r **into** $a[s_1, o]$;
end



SPM and Multiparent Create

- *cc* extended in obvious way
  - $cc \subseteq TS \times \ldots \times TS \times T$
- Symbols
  - $\mathbf{X}_1, \ldots, \mathbf{X}_n$ parents, $\mathbf{Y}$ created
  - $R_{1,i}, R_{2,i}, R_3, R_{4,i} \subseteq R$
- Rules
  - $cr_{\mathrm{P},i}(\tau(\mathbf{X}_1), \ldots, \tau(\mathbf{X}_n)) = \mathbf{Y}/R_{1,1} \cup \mathbf{X}_i/R_{2,i}$
  - $cr_{\mathrm{C}}(\tau(\mathbf{X}_1), \ldots, \tau(\mathbf{X}_n)) = \mathbf{Y}/R_3 \cup \mathbf{X}_1/R_{4,1} \cup \ldots \cup \mathbf{X}_n/R_{4,n}$



Example

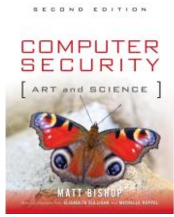
- Anna, Bill must do something cooperatively
  - But they don't trust each other
- Jointly create a proxy
  - Each gives proxy only necessary rights
- In ESPM:
  - Anna, Bill type a; proxy type p; right $x \in R$
  - $cc(a, a) = p$
  - $cr_{\text{Anna}}(a, a, p) = cr_{\text{Bill}}(a, a, p) = \varnothing$
  - $cr_{\text{proxy}}(a, a, p) = \{ \text{Anna}/x, \text{Bill}//x \}$

2-Parent Joint Create Suffices

- Goal: emulate 3-parent joint create with 2-parent joint create
- Definition of 3-parent joint create (subjects $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$; child $\mathbf{C}$):
  - $cc(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = Z \subseteq T$
  - $cr_{\mathbf{P}1}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{1,1} \cup \mathbf{P}_1/R_{2,1}$
  - $cr_{\mathbf{P}2}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{2,1} \cup \mathbf{P}_2/R_{2,2}$
  - $cr_{\mathbf{P}3}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{3,1} \cup \mathbf{P}_3/R_{2,3}$

General Approach

- Define agents for parents and child
  - Agents act as surrogates for parents
  - If create fails, parents have no extra rights
  - If create succeeds, parents, child have exactly same rights as in 3-parent creates
    - Only extra rights are to agents (which are never used again, and so these rights are irrelevant)



Entities and Types

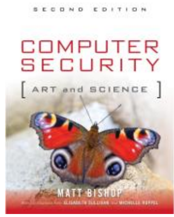
- Parents **P**$_1$, **P**$_2$, **P**$_3$ have types p_1, p_2, p_3
- Child **C** of type c
- Parent agents **A**$_1$, **A**$_2$, **A**$_3$ of types a_1, a_2, a_3
- Child agent **S** of type s
- Type t is parentage
  - if **X**/$t \in dom$(**Y**), **X** is **Y**'s parent
- Types t, a_1, a_2, a_3, s are new types

can•create

- Following added to *can•create*:
  - $\mathrm{cc}(p_1) = a_1$
  - $\mathrm{cc}(p_2, a_1) = a_2$
  - $\mathrm{cc}(p_3, a_2) = a_3$
    - Parents creating their agents; note agents have maximum of 2 parents
  - $\mathrm{cc}(a_3) = s$
    - Agent of all parents creates agent of child
  - $\mathrm{cc}(s) = c$
    - Agent of child creates child

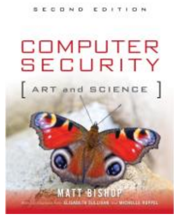
Creation Rules

- Following added to create rule:
  - $cr_P(p_1, a_1) = \varnothing$
  - $cr_C(p_1, a_1) = p_1/Rtc$
    - Agent's parent set to creating parent; agent has all rights over parent
  - $cr_{Pfirst}(p_2, a_1, a_2) = \varnothing$
  - $cr_{Psecond}(p_2, a_1, a_2) = \varnothing$
  - $cr_C(p_2, a_1, a_2) = p_2/Rtc \cup a_1/tc$
    - Agent's parent set to creating parent and agent; agent has all rights over parent (but not over agent)



Creation Rules

- $cr_{Pfirst}(p_3, a_2, a_3) = \varnothing$
- $cr_{Psecond}(p_3, a_2, a_3) = \varnothing$
- $cr_C(p_3, a_2, a_3) = p_3/Rtc \cup a_2/tc$
    - Agent's parent set to creating parent and agent; agent has all rights over parent (but not over agent)
- $cr_P(a_3, s) = \varnothing$
- $cr_C(a_3, s) = a_3/tc$
    - Child's agent has third agent as parent $cr_P(a_3, s) = \varnothing$
- $cr_P(s, c) = \mathbf{C}/Rtc$
- $cr_C(s, c) = c/R_3t$
    - Child's agent gets full rights over child; child gets R_3 rights over agent



Link Predicates

- Idea: no tickets to parents until child created
  - Done by requiring each agent to have its own parent rights
  - $link_1(\mathbf{A}_2, \mathbf{A}_1) = \mathbf{A}_1/t \in dom(\mathbf{A}_2) \land \mathbf{A}_2/t \in dom(\mathbf{A}_2)$
  - $link_1(\mathbf{A}_3, \mathbf{A}_2) = \mathbf{A}_2/t \in dom(\mathbf{A}_3) \land \mathbf{A}_3/t \in dom(\mathbf{A}_3)$
  - $link_2(\mathbf{S}, \mathbf{A}_3) = \mathbf{A}_3/t \in dom(\mathbf{S}) \land \mathbf{C}/t \in dom(\mathbf{C})$
  - $link_3(\mathbf{A}_1, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_1)$
  - $link_3(\mathbf{A}_2, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_2)$
  - $link_3(\mathbf{A}_3, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_3)$
  - $link_4(\mathbf{A}_1, \mathbf{P}_1) = \mathbf{P}_1/t \in dom(\mathbf{A}_1) \land \mathbf{A}_1/t \in dom(\mathbf{A}_1)$
  - $link_4(\mathbf{A}_2, \mathbf{P}_2) = \mathbf{P}_2/t \in dom(\mathbf{A}_2) \land \mathbf{A}_2/t \in dom(\mathbf{A}_2)$
  - $link_4(\mathbf{A}_3, \mathbf{P}_3) = \mathbf{P}_3/t \in dom(\mathbf{A}_3) \land \mathbf{A}_3/t \in dom(\mathbf{A}_3)$

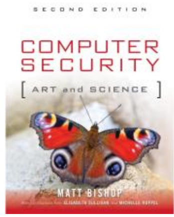
Filter Functions

- $f_1(a_2, a_1) = a_1/t \cup c/Rtc$
- $f_1(a_3, a_2) = a_2/t \cup c/Rtc$
- $f_2(s, a_3) = a_3/t \cup c/Rtc$
- $f_3(a_1, c) = p_1/R_{4,1}$
- $f_3(a_2, c) = p_2/R_{4,2}$
- $f_3(a_3, c) = p_3/R_{4,3}$
- $f_4(a_1, p_1) = c/R_{1,1} \cup p_1/R_{2,1}$
- $f_4(a_2, p_2) = c/R_{1,2} \cup p_2/R_{2,2}$
- $f_4(a_3, p_3) = c/R_{1,3} \cup p_3/R_{2,3}$

Construction

Create $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, **S**, **C**; then

- $\mathbf{P}_1$ has no relevant tickets
- $\mathbf{P}_2$ has no relevant tickets
- $\mathbf{P}_3$ has no relevant tickets
- $\mathbf{A}_1$ has $\mathbf{P}_1/Rtc$
- $\mathbf{A}_2$ has $\mathbf{P}_2/Rtc \cup \mathbf{A}_1/tc$
- $\mathbf{A}_3$ has $\mathbf{P}_3/Rtc \cup \mathbf{A}_2/tc$
- **S** has $\mathbf{A}_3/tc \cup \mathbf{C}/Rtc$
- **C** has $\mathbf{C}/R_3t$



Construction

- Only $link_2(\mathbf{S}, \mathbf{A}_3)$ true $\Rightarrow$ apply f_2
  - $\mathbf{A}_3$ has $\mathbf{P}_3/Rtc \cup \mathbf{A}_2/t \cup \mathbf{A}_3/t \cup \mathbf{C}/Rtc$
- Now $link_1(\mathbf{A}_3, \mathbf{A}_2)$ true $\Rightarrow$ apply f_1
  - $\mathbf{A}_2$ has $\mathbf{P}_2/Rtc \cup \mathbf{A}_1/tc \cup \mathbf{A}_2/t \cup \mathbf{C}/Rtc$
- Now $link_1(\mathbf{A}_2, \mathbf{A}_1)$ true $\Rightarrow$ apply f_1
  - $\mathbf{A}_1$ has $\mathbf{P}_2/Rtc \cup \mathbf{A}_1/t \cup \mathbf{C}/Rtc$
- Now all $link_3$s true $\Rightarrow$ apply f_3
  - $\mathbf{C}$ has $\mathbf{C}/R_3 \cup \mathbf{P}_1/R_{4,1} \cup \mathbf{P}_2/R_{4,2} \cup \mathbf{P}_3/R_{4,3}$

Finish Construction

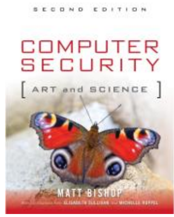
- Now $link_4$ is true $\Rightarrow$ apply f_4
  - $\mathbf{P}_1$ has $\mathbf{C}/R_{1,1} \cup \mathbf{P}_1/R_{2,1}$
  - $\mathbf{P}_2$ has $\mathbf{C}/R_{1,2} \cup \mathbf{P}_2/R_{2,2}$
  - $\mathbf{P}_3$ has $\mathbf{C}/R_{1,3} \cup \mathbf{P}_3/R_{2,3}$
- 3-parent joint create gives same rights to $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, $\mathbf{C}$
- If create of **C** fails, $link_2$ fails, so construction fails

Theorem

- The two-parent joint creation operation can implement an *n*-parent joint creation operation with a fixed number of additional types and rights, and augmentations to the link predicates and filter functions.
- **Proof**: by construction, as above
  - Difference is that the two systems need not start at the same initial state

Theorems

- Monotonic ESPM and the monotonic HRU model are equivalent.
- Safety question in ESPM also decidable if acyclic attenuating scheme
  - Proof similar to that for SPM

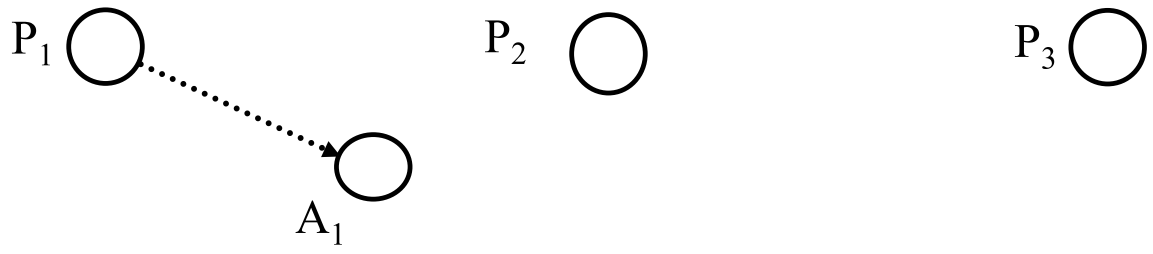


Expressiveness

- Graph-based representation to compare models
- Graph
  - Vertex: represents entity, has static type
  - Edge: represents right, has static type
- Graph rewriting rules:
  - *Initial state operations* create graph in a particular state
  - *Node creation operations* add nodes, incoming edges
  - *Edge adding operations* add new edges between existing vertices

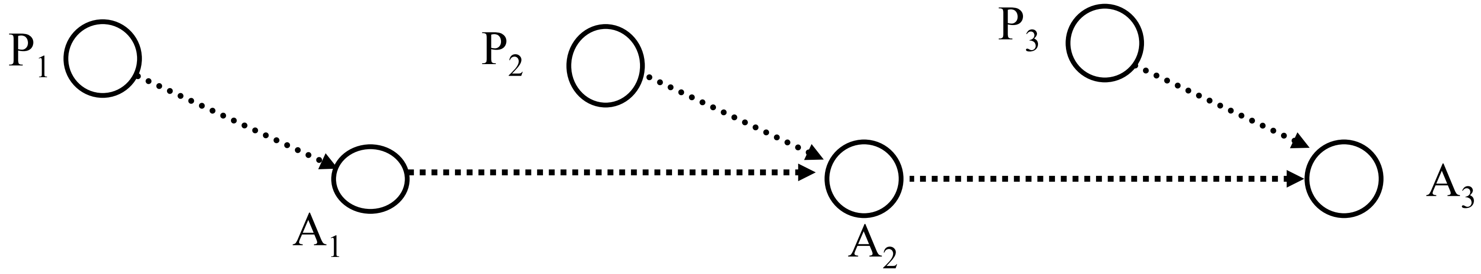
Example: 3-Parent Joint Creation

- Simulate with 2-parent
  - Nodes $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ parents
  - Create node **C** with type *c* with edges of type *e*
  - Add node $\mathbf{A}_1$ of type *a* and edge from $\mathbf{P}_1$ to $\mathbf{A}_1$ of type *e*´



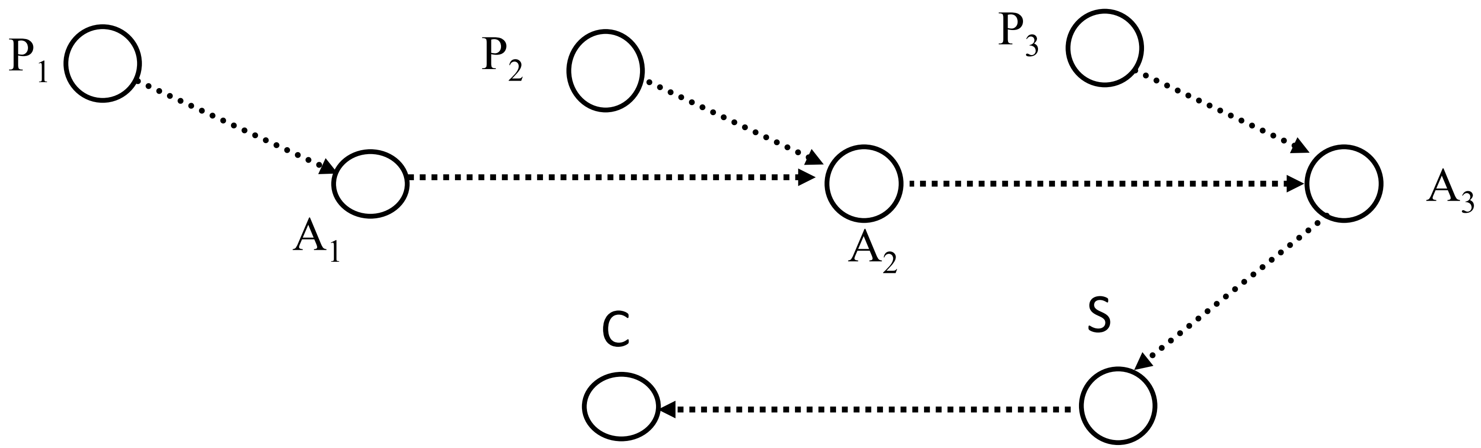
Next Step

- **A_1**, **P_2** create **A_2**; **A_2**, **P_3** create **A_3**
- Type of nodes, edges are *a* and *e*´



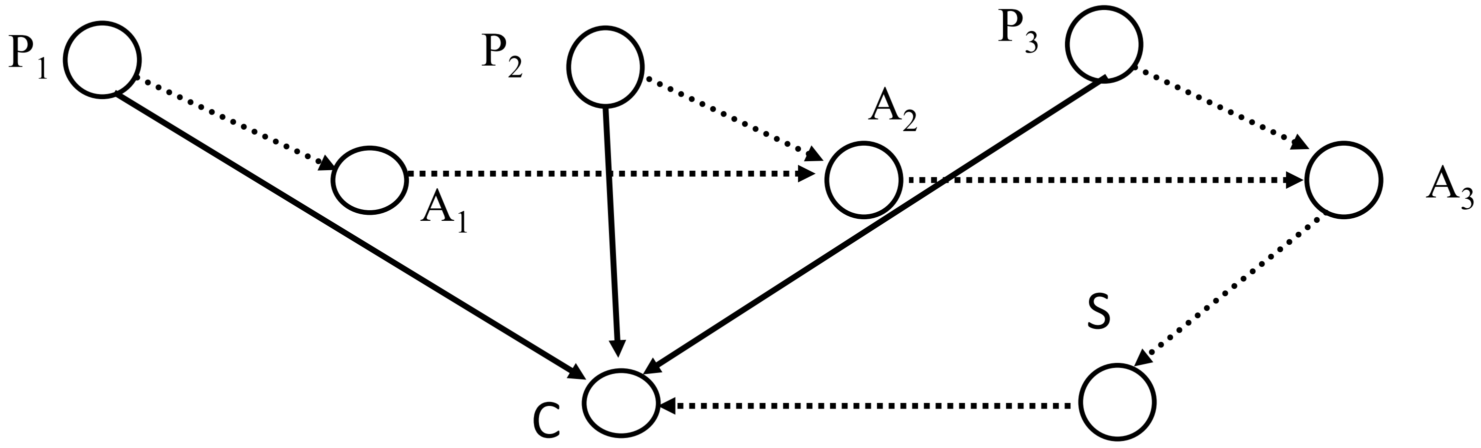
Next Step

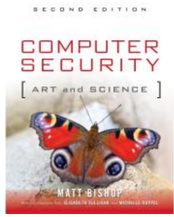
- **A_3** creates **S**, of type *a*
- **S** creates **C**, of type *c*



Last Step

- Edge adding operations:
  - **$P_1 \rightarrow A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow S \rightarrow C$**: **$P_1$** to **C** edge type *e*
  - **$P_2 \rightarrow A_2 \rightarrow A_3 \rightarrow S \rightarrow C$**: **$P_2$** to **C** edge type *e*
  - **$P_3 \rightarrow A_3 \rightarrow S \rightarrow C$**: **$P_3$** to **C** edge type *e*





Definitions

- *Scheme*: graph representation as above
- *Model*: set of schemes
- Schemes *A*, *B* *correspond* if graph for both is identical when all nodes with types not in *A* and edges with types in *A* are deleted

Example

- Above 2-parent joint creation simulation in scheme *TWO*
- Equivalent to 3-parent joint creation scheme *THREE* in which $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, **C** are of same type as in *TWO*, and edges from $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ to **C** are of type *e*, and no types *a* and *e*´ exist in *TWO*

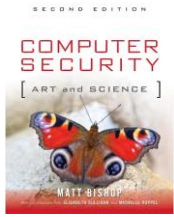
Simulation

Scheme *A* simulates scheme *B* iff

- every state *B* can reach has a corresponding state in *A* that *A* can reach; and
- every state that *A* can reach either corresponds to a state *B* can reach, or has a successor state that corresponds to a state *B* can reach
  - The last means that *A* can have intermediate states not corresponding to states in *B*, like the intermediate ones in *TWO* in the simulation of *THREE*

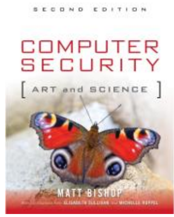
Expressive Power

- If there is a scheme in *MA* that no scheme in *MB* can simulate, *MB less expressive than MA*
- If every scheme in *MA* can be simulated by a scheme in *MB*, *MB as expressive as MA*
- If *MA* as expressive as *MB* and *vice versa*, *MA* and *MB equivalent*



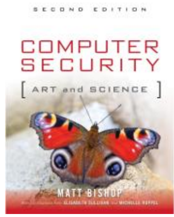
Example

- Scheme *A* in model *M*
  - Nodes $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$
  - 2-parent joint create
  - 1 node type, 1 edge type
  - No edge adding operations
  - Initial state: $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, no edges
- Scheme *B* in model *N*
  - All same as *A* except no 2-parent joint create
  - 1-parent create
- Which is more expressive?



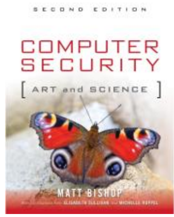
Can *A* Simulate *B*?

- Scheme *A* simulates 1-parent create: have both parents be same node
  - Model *M* as expressive as model *N*



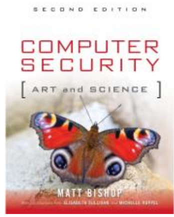
Can *B* Simulate *A*?

- Suppose $\mathbf{X}_1$, $\mathbf{X}_2$ jointly create **Y** in *A*
  - Edges from $\mathbf{X}_1$, $\mathbf{X}_2$ to **Y**, no edge from $\mathbf{X}_3$ to **Y**
- Can *B* simulate this?
  - Without loss of generality, $\mathbf{X}_1$ creates **Y**
  - Must have edge adding operation to add edge from $\mathbf{X}_2$ to **Y**
  - One type of node, one type of edge, so operation can add edge between any 2 nodes



No

- All nodes in *A* have even number of incoming edges
  - 2-parent create adds 2 incoming edges
- Edge adding operation in *B* that can edge from $\mathbf{X}_2$ to **C** can add one from $\mathbf{X}_3$ to **C**
  - *A* cannot enter this state
  - *B* cannot transition to a state in which **Y** has even number of incoming edges
    - No remove rule
- So *B* cannot simulate *A*; *N* less expressive than *M*

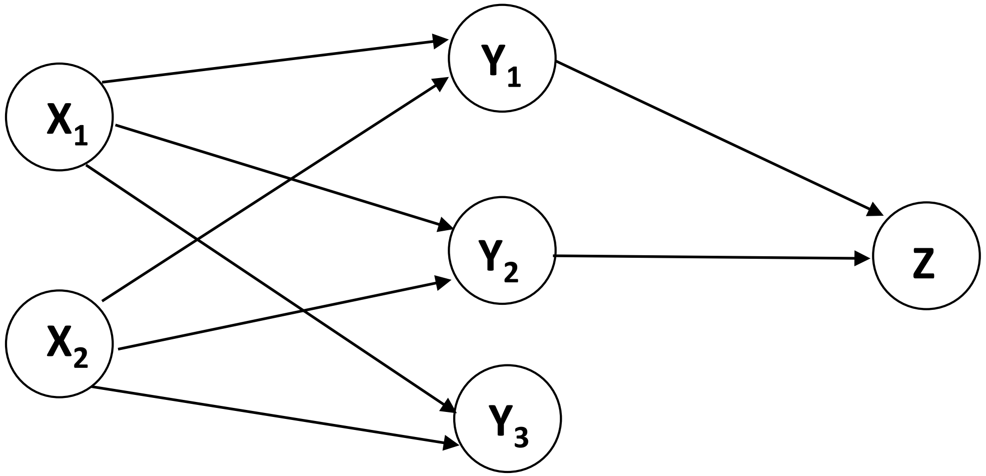


Theorem

- Monotonic single-parent models are less expressive than monotonic multiparent models
- Proof by contradiction
  - Scheme *A* is multiparent model
  - Scheme *B* is single parent create
  - Claim: *B* can simulate *A*, without assumption that they start in the same initial state
    - Note: example assumed same initial state

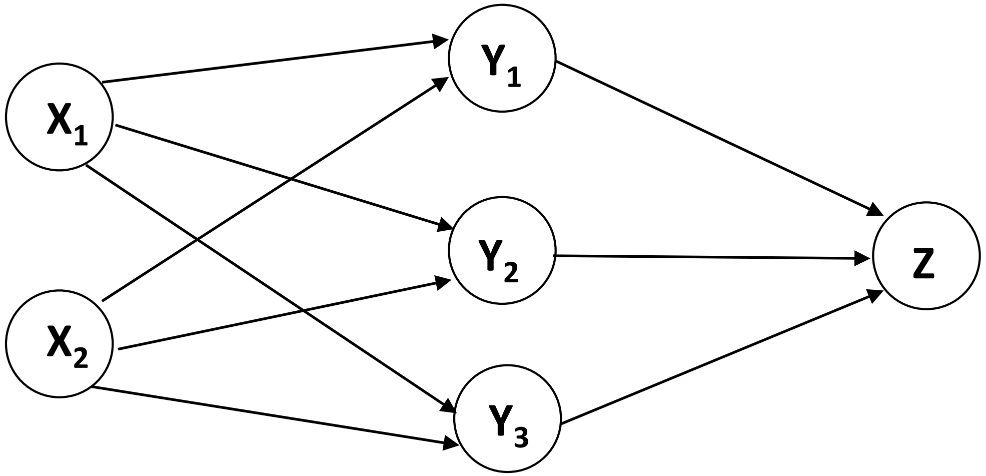
Outline of Proof

- $\mathbf{X}_1$, $\mathbf{X}_2$ nodes in *A*
  - They create $\mathbf{Y}_1$, $\mathbf{Y}_2$, $\mathbf{Y}_3$ using multiparent create rule
  - $\mathbf{Y}_1$, $\mathbf{Y}_2$ create $\mathbf{Z}$, again using multiparent create rule
  - *Note*: no edge from $\mathbf{Y}_3$ to $\mathbf{Z}$ can be added, as *A* has no edge-adding operation



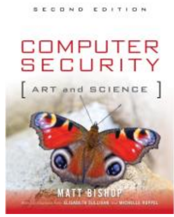
Outline of Proof

- **W**, $\mathbf{X}_1$, $\mathbf{X}_2$ nodes in *B*
    - **W** creates $\mathbf{Y}_1$, $\mathbf{Y}_2$, $\mathbf{Y}_3$ using single parent create rule, and adds edges for $\mathbf{X}_1$, $\mathbf{X}_2$ to all using edge adding rule
    - $\mathbf{Y}_1$ creates **Z**, again using single parent create rule; now must add edge from $\mathbf{Y}_2$ to **Z** to simulate *A*
    - Use same edge adding rule to add edge from $\mathbf{Y}_3$ to **Z**: cannot duplicate this in scheme *A*!



Meaning

- Scheme *B* cannot simulate scheme *A*, contradicting hypothesis
- ESPM more expressive than SPM
  - ESPM multiparent and monotonic
  - SPM monotonic but single parent



Typed Access Matrix Model

- Like ACM, but with set of types T
  - All subjects, objects have types
  - Set of types for subjects TS
- Protection state is (S, O, τ, A)
  - τ:$O \rightarrow T$ specifies type of each object
  - If **X** subject, τ(**X**) in TS
  - If **X** object, τ(**X**) in $T - TS$

Create Rules

- Subject creation
  - **create subject** *s* **of type** *ts*
  - *s* must not exist as subject or object when operation executed
  - $ts \in TS$
- Object creation
  - **create object** *o* **of type** *to*
  - *o* must not exist as subject or object when operation executed
  - $to \in T - TS$

Create Subject

- Precondition: $s \notin S$
- Primitive command: **create subject** *s* **of type** *t*
- Postconditions:
  - $S' = S \cup \{ s \}, O' = O \cup \{ s \}$
  - $(\forall y \in O)[\tau'(y) = \tau(y)], \tau'(s) = t$
  - $(\forall y \in O')[a'[s, y] = \varnothing], (\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Create Object

- Precondition: $o \notin O$
- Primitive command: **create object** o **of type** t
- Postconditions:
  - $S' = S$, $O' = O \cup \{ o \}$
  - $(\forall y \in O)[\tau'(y) = \tau(y)]$, $\tau'(o) = t$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Definitions

- MTAM Model: TAM model without **delete**, **destroy**
  - MTAM is Monotonic TAM
- $\alpha(x_1{:}t_1, \ldots, x_n{:}t_n)$ create command
  - t_i child type in α if any of **create subject** x_i **of type** t_i or **create object** x_i **of type** t_i occur in α
  - t_i parent type otherwise

Cyclic Creates

command $cry{\bullet}havoc(s_1 : u,\ s_2 : u,\ o_1 : v,\ o_2 : v,\ o_3 : w,\ o_4 : w)$

  create subject s_1 **of type** u;

  create object o_1 **of type** v;

  create object o_3 **of type** w;

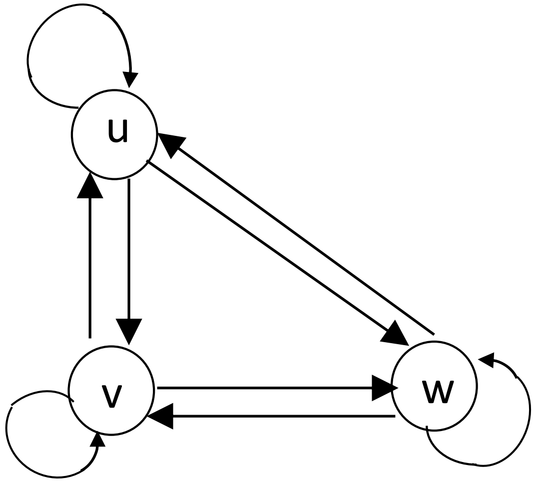
  enter r **into** $a[s_2, s_1]$;

  enter r **into** $a[s_2, o_2]$;

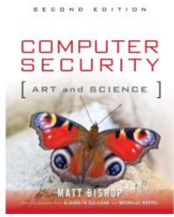
  enter r **into** $a[s_2, o_4]$

end

Creation Graph



- *u*, *v*, *w* child types
- *u*, *v*, *w* also parent types
- Graph: lines from parent types to child types
- This one has cycles



Acyclic Creates

command $cry{\bullet}havoc(s_1 : u,\ s_2 : u,\ o_1 : v,\ o_3 : w)$

  create object o_1 **of type** v;

  create object o_3 **of type** w;

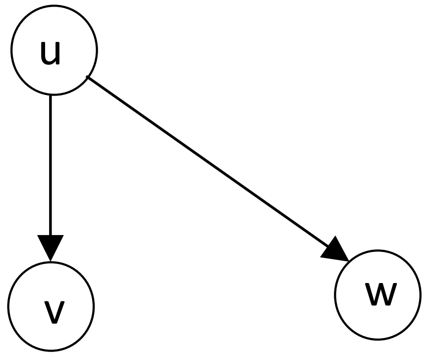
  enter r **into** $a[s_2, s_1]$;

  enter r **into** $a[s_2, o_1]$;

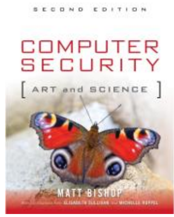
  enter r **into** $a[s_2, o_3]$

end

Creation Graph

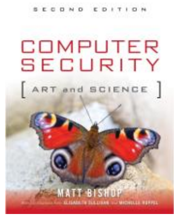


- *v*, *w* child types
- *u* parent type
- Graph: lines from parent types to child types
- This one has no cycles



Theorems

- Safety decidable for systems with acyclic MTAM schemes
  - In fact, it’s *NP-hard*
- Safety for acyclic ternary MATM decidable in time polynomial in the size of initial ACM
  - “Ternary” means commands have no more than 3 parameters
  - Equivalent in expressive power to MTAM

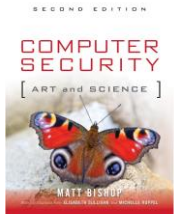


Security Properties

- Question: given two models, do they have the same security properties?
  - First comes theory
  - Then comes an example comparison
- Basic idea: view access request as query asking if subject has right to perform action on object

Alternate Definition of "Scheme"

- Σ set of states
- *Q* set of queries
- $e: \Sigma \times Q \rightarrow \{true, false\}$
    - Called *entailment relation*
- *T* set of state transition rules
- (Σ, Q, e, T) is an *access control scheme*



Alternate Definition of “Scheme”

- *s* tries to access *o*
    - Corresponds to query $q \in Q$
- If state $\sigma \in \Sigma$ allows access, then $e(\sigma, q) = true$; otherwise, $e(\sigma, q) = false$
- Write change of state from σ_0 to σ_1 as $\sigma_0 \mapsto \sigma_1$
    - Emphasizing we’re looking at *permissions*
    - Multiple transitions are $\sigma_0 \mapsto_\tau^* \sigma_n$
        - Σ_n said to be *τ-reachable from* σ_0

Example: Take-Grant

- Σ set of all possible protection graphs
- *Q* set of queries

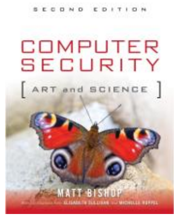
  $\{ \mathit{can{\bullet}share}(\alpha, \mathbf{v}_1, \mathbf{v}_2, G_0) \mid \alpha \in R, \mathbf{v}_1, \mathbf{v}_2 \in G_0 \}$
- $e(\sigma_0, q) = \mathit{true}$ if q holds; $e(\sigma_0, q) = \mathit{false}$ if not
- *T* set of sequences of take, grant, create, remove rules

Security Analysis Instance

- Let (Σ, *Q*, *e*, *T*) be an *access control scheme*
- Tuple (σ, *q*, τ, Π) is *security analysis instance*, where:
  - $\sigma \in \Sigma$
  - $q \in Q$
  - $\tau \in T$
  - Π is ∀ or ∃
- If Π is ∃, *existential* security analysis
  - Is there a state σ′ such that $\sigma \mapsto_{\tau}^{*} \sigma'$, $e(\sigma', q) = \mathit{true}$?
- If Π is ∀, *universal* security analysis
  - For all states σ′ such that $\sigma \mapsto_{\tau}^{*} \sigma'$, is $e(\sigma', q) = \mathit{true}$?

Example: Take-Grant

- $\sigma_0 = G_0$
- q is *can•share*(r, $\mathbf{v}_1$, $\mathbf{v}_2$, G_0)
- τ is sequence of take-grant rules
- Π is ∃
- Security analysis instance examines whether $\mathbf{v}_1$ has r rights over $\mathbf{v}_2$ in graph with initial state G_0
- So safety question is security analysis instance



Comparing Two Models

- Each query in A corresponds to a query in B
- Each (state, state transition) in A corresponds to (state, state transition) in B

Formally:

- $A = (\Sigma^A, Q^A, e^A, T^A)$ and $B = (\Sigma^B, Q^B, e^B, T^B)$
- *mapping* from A to B is:
  - $f : (\Sigma^A \times T^A) \cup Q^A \rightarrow (\Sigma^B \times T^B) \cup Q^B$

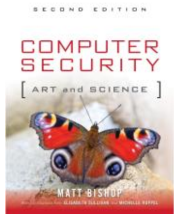
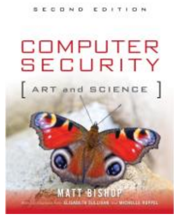


Image of Instance

- f mapping from A to B
- *image of a security analysis instance* $(\sigma^A, q^A, \tau^A, \Pi)$ under f is $(\sigma^B, q^B, \tau^B, \Pi)$, where:
  - $f((\sigma^A, \tau^A)) = (\sigma^B, \tau^B)$
  - $f(q^A) = q^B$
- f is *security-preserving* if every security analysis instance in A is true iff its image is true



Composition of Queries

- Let (Σ, *Q*, *e*, *T*) be an *access control scheme*
- Tuple (σ, *φ*, τ, Π) is compositional *security analysis instance*, where *φ* is propositional logic formula of queries from *Q*
- *image of compositional security analysis instance* defined similarly to previous
- *f* is *strongly security-preserving* if every compositional security analysis instance in *A* is true iff its image is true

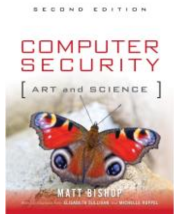
State-Matching Reduction

- $A = (\Sigma^A, Q^A, e^A, T^A)$, $B = (\Sigma^B, Q^B, e^B, T^B)$, f mapping from A to B
- σ^A, σ^B equivalent under the mapping f when
  - $e^A(\sigma^A, q^A) = e^B(\sigma^B, q^B)$
- f state-matching reduction if for all $\sigma^A \in S^A$, $\tau^A \in T^A$,

  $(\sigma^B, \tau^B) = f((\sigma^A, \tau^A))$ has the following properties:

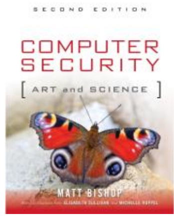
Property 1

- For every state σ'^{A} in scheme A such that $\sigma^{A} \mapsto_{\tau}^{*} \sigma'^{A}$, there is a state σ'^{B} in scheme B such that $\sigma^{B} \mapsto_{\tau}^{*} \sigma'^{B}$, and σ'^{A} and σ'^{B} are equivalent under the mapping f
    - That is, for every reachable state in *A*, a matching state in *B* gives the same answer for every query



Property 2

- For every state σ'^{B} in scheme *B* such that $\sigma^{B} \mapsto_{\tau}^{*} \sigma'^{B}$, there is a state σ'^{A} in scheme *A* such that $\sigma^{A} \mapsto_{\tau}^{*} \sigma'^{A}$, and σ'^{A} and σ'^{B} are equivalent under the mapping *f*
  - That is, for every reachable state in *B*, a matching state in *A* gives the same answer for every query



Theorem

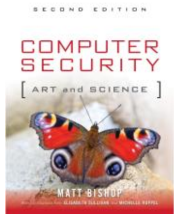
Mapping *f* from scheme *A* to *B* is strongly security-preserving iff *f* is a state-matching reduction

Proof ($\Rightarrow$)

- Must show (σ^A, φ^A, τ^A, Π) true iff (σ^B, φ^B, τ^B, Π) true
- Π is ∃: assume τ^A-reachable state σ'^A from σ^A in which φ^A true
  - By property 1, there is a state σ'^B corresponding to σ'^A in which φ^B holds
- Π is ∀: assume τ^A-reachable state σ'^A from σ^A in which φ^A false
  - By property 1, there is a state σ'^B corresponding to σ'^A in which φ^B false
- Same for φ^B with τ^B-reachable state σ'^B from σ^B
- So (σ^A, φ^A, τ^A, Π) true iff (σ^B, φ^B, τ^B, Π) true

Proof ($\Longleftarrow$)

- Let f be map from A to B but not state-matching reduction. Then there are $\sigma^A \in S^A$, $\tau^A \in T^A$, $(\sigma^B, \tau^B) = f((\sigma^A, \tau^A))$ violating at least one of the properties
- Assume it's property 1; σ^A, σ^B corresponding states. There is a τ^A-reachable state σ'^A from σ^A such that no τ^B-reachable state from σ^B is equivalent to σ'^B
- Generate φ^A and φ^B such that the existential compositional security analysis in A is true but in B is false
    - To do this, look at each $q^A \in Q^A$
    - If $e(\sigma'^A, q^A) = \mathit{true}$, conjoin q^A to φ^A; otherwise, conjoin $\neg q^A$ to φ^A
    - Then $e(\sigma'^A, q^A) = \mathit{true}$ but for $\varphi^B = f(\varphi^A)$ and all states σ'^B that are τ^B-reachable from σ^B, $e(\sigma'^B, q^B) = \mathit{false}$
- Thus, f is not strongly security-preserving
- Argument for property 2 is similar



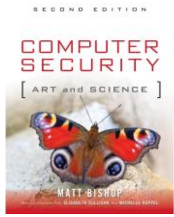
Expressive Power

If access control model *MA* has a scheme that cannot be mapped into a scheme in access control model *MB* using a state-matching reduction, then model *MB* is *less expressive than* model *MA*.

If every scheme in model *MA* can be mapped into a scheme in model *MB* using a state-matching reduction, then model *MB* is *as expressive as* model *MA*.

If *MA* is as expressive as *MB*, and MB is as expressive as *MA*, the models are *equivalent*

- Note this does not assume monotonicity, unlike earlier definition



Augmented Typed Access Control Matrix

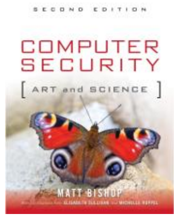
- Add a test for the *absence* of rights to TAM

```
command add•right(s:u, o:v)
    if own in a[s,o] and r not in a[s,o]
    then
        enter r into a[s,o]
end
```

- How does this affect the answer to the safety question?

Safety Question

- ATAM can be mapped onto TAM
- But will the mapping, or any such mapping, preserve security properties?
- Approach: consider TAM as an access control model



TAM as Access Control Model

- S set of subjects; S_σ subjects in state σ
- O set of objects; O_σ objects in state σ
- R set of rights; R_σ rights in state σ
- T set of types; T_σ subjects in state σ
- $t : S_\sigma \cup O_\sigma \longrightarrow T_\sigma$ gives type of any subject or object
- State σ defined as $(S_\sigma, O_\sigma, R_\sigma, T_\sigma, t)$
- In TAM, query is of form "is $r \in a[s,o]$", and $e(s, r \in a[s,o])$ true iff $s \in S_\sigma$, $o \in O_\sigma$, $r \in R_\sigma$, $r \in a_\sigma[s,o]$ are true

ATAM as Access Control Model

Same as TAM with one addition:

- ATAM also allows queries of form "is $r \notin a[s,o]$", and $e(s, r \notin a[s,o])$ true iff $s \in S_\sigma$, $o \in O_\sigma$, $r \in R_\sigma$, $r \notin a_\sigma[s,o]$ are true

Theorem

A state-matching reduction from ATAM to Tam does not exist.

Outline of proof: by contradiction

- Consider two state transitions, one that creates subject and one that adds right *r* to an element of the matrix
- Can determine an upper bound on the number of answers to TAM query a command can change; depends on state and commands

Proof

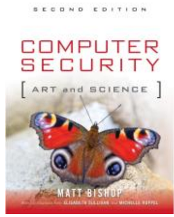
- Assume *f* is state-matching reduction from ATAM to TAM
- Consider simple ATAM scheme:
  - Initial state σ_0 has no subjects, objects
  - All entities have type *t*
  - Only one right *r*
  - Query $q_{ij} = r \in a[s,o]$; query $\underline{q_{ij}} = r \notin a[s,o]$
  - 2 state transition rules
    - *make•subj*(s : t) creates subject *s* of type *t*
    - *add•right*(x : t, y : t) adds right *r* to $a[x, y]$

Proof

- TAM: superscript *T* represents components of that system
  - So initial state is $\sigma_0^T = f(\sigma_0)$, transitions are $\tau^T = f(\tau)$
- By definition of state-matching reduction, how *f* maps queries does not depend on initial state or state transitions of a model
- Let *p*, *q* be queries in ATAM and p^T, q^T the corresponding queries in TAM; if $p \neq q$, then $p^T \neq q^T$
- As commands in TAM execute, they can change the value (response) of q_{ij}
- Upper bound on the number of values of queries a single command can change is *m* (number of **enter** or **add•right** operations)

Proof

- Choose $n > m$
- In ATAM, construct state σ_k such that:
  - $\sigma_0 \longrightarrow^* \sigma_k$; and
  - $e(\sigma_k, \neg q_{1,1} \wedge \underline{q_{1,1}} \wedge \ldots \wedge \neg q_{n,n} \wedge \underline{q_{n,n}})$ is true
- So $e(\sigma_k, q_{i,j})$ is false, $e(\sigma_k, \underline{q_{i,j}})$ is true for all $1 \leq i, j \leq n$
- As f is a state-matching reduction, there is a state σ_k^T in TAM that causes the corresponding queries to be answered the same way
- Consider $\sigma_0^T \longrightarrow \sigma_1^T \longrightarrow \ldots \longrightarrow \sigma_k^T$; choose first state σ_C^T such that $e(\sigma_C^T, q_{i,j}^T \vee \underline{q_{i,j}^T})$ is true for all $1 \leq i, j \leq n$



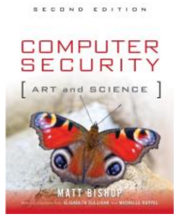
Proof

- In σ_{C-1}^T, $e(\sigma_{C-1}^T, q_{v,w}^T \lor \underline{q_{v,w}}^T)$ is false for some $1 \le v, w \le n$, so $e(\sigma_{C-1}^T, \neg q_{v,w}^T \land \neg \underline{q_{v,w}}^T)$ is true
- State σ in ATAM for which $e(\sigma, \neg q_{v,w} \land \neg \underline{q_{v,w}})$ is true is one in which either s_v or s_w or both does not exist
- Thus in that state, one of the following 2 queries holds:
  - $Q_1 = \neg q_{v,1} \land \neg \underline{q_{v,1}} \land \ldots \land \neg q_{n,v} \land \neg \underline{q_{n,v}}$
  - $Q_1 = \neg q_{w,1} \land \neg \underline{q_{w,1}} \land \ldots \land \neg q_{n,w} \land \neg \underline{q_{n,w}}$
- So in TAM, $e(\sigma_{C-1}^T, Q_1^T \land Q_2^T)$ is true

Proof

- Now consider the transition from σ_{C-1}^T to σ_C^T
- Values of at least n queries in Q_1 or Q_2 must change from false to true
- But each command can change at most $m < n$ queries
- This is a contradiction
- So no such f can exist, proving the result

Thus, ATAM can express security properties that TAM cannot



Key Points

- Safety problem undecidable
- Limiting scope of systems can make problem decidable
- Types critical to safety problem’s analysis