CS52600: Information Security
Protection Models

September 1, 2010
Prof. Chris Clifton

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

Jones, Lipton, Snyder FOCS'76

3

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'

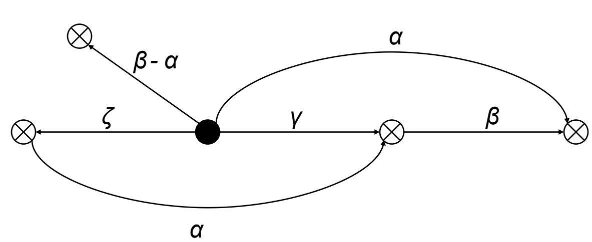
$R = \{ t, g, r, w, \dots \}$ set of rights

4

Take-Grant Protection Model

- System is directed graph
  - Subject: ●
  - Object: ○
  - (labeled) edge: {rights}

  Both: ⊗
- Take rule: if $t \in \gamma$, $\alpha \subseteq \beta$, can add transitive edge
- Grant rule: if $g \in \zeta$, $\alpha \subseteq \gamma$, can add (grant) edge between recipients
- Create, Remove rules



5

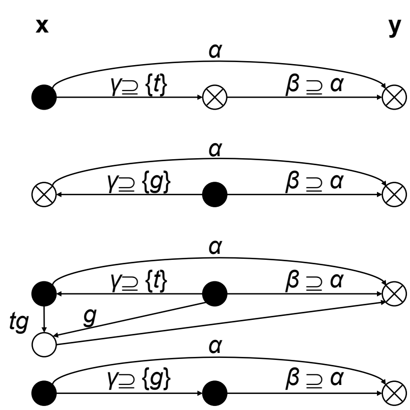
Take-Grant Protection Model: Sharing

- Given G_0, can vertex **x** obtain α rights over **y**?
  - Can_share(α,**x**,**y**,G_0) iff $G_0 \vdash^* G_n$ using the above rules and α edge from **x** to **y** in G_n
- *tg-path*: $\mathbf{v}_0,\ldots,\mathbf{v}_n$ where t or g edge between any $\mathbf{v}_i$, $\mathbf{v}_{i+1}$
  - Vertices *tg-connected* if *tg-path* between them
- Theorem: Any two subjects with *tg-path* of length 1 can cause rights to be shared

6

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

Can_share(α,**x**,**y**,G_0)

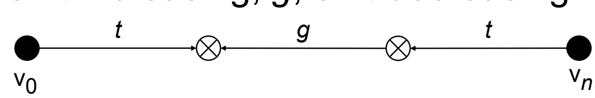


- Four possible length 1 tg-paths
- Take rule
- Grant rule
- Sequence:
  - Create
  - Take
  - Grant
  - Take

7

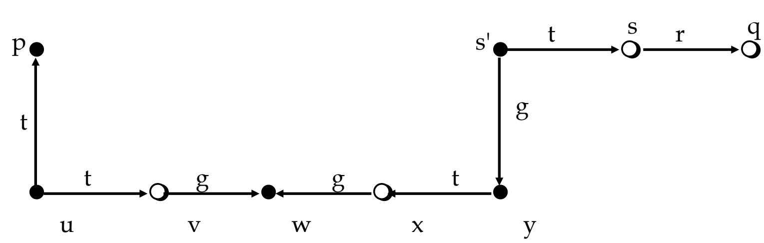
Other definitions

- Island: Maximal *tg*-connected subject-only subgraph
  - Can_share all rights in island
  - Proof: Induction from previous theorem
- Bridge: *tg*-path between subjects v_0 and v_n with edges of the following form:
  - All *t*
  - 0+ *t* increasing, *g*, 0+ *t* decreasing



8

Example

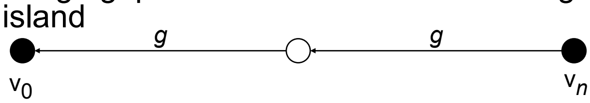


- islands { p, u } { w } { y, s' }
- bridges u, v, w; w, x, y
- initial span p (associated word ν)
- terminal span s's (associated word t)

9

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

- Can_share(α,**x**,**y**,G_0) if **x** and **y** are subjects and there is an α edge from **x** to **y** in G_0 or if:
  - ∃ a subject s ∈ G_0 with an **s** to **y** α edge, and
  - ∃ islands I_1, …, I_n such that **x** ∈ I_1, **s** ∈ I_n, and there is a bridge from I_j to I_{j+1}
- Proof: Islands above, bridge – take in both directions to grant link, then one takes "grant" and grants to other
- If **x** and **y** are subjects, "only if" holds
  - If no take/grant or two grants between objects, can't bridge gap. Otherwise it is either a bridge or an island



10

What about objects?

- **x** *initially spans* to **y** if **x** is a subject and there is a *tg*-path between them with *t* edges ending in a *g* edge
  - **x** can grant a right to **y**
- **x** terminally spans to **y** if **x** is a subject and there is a *tg*-path between them with *t* edges
  - **x** can take a right from **y**

11

Theorem: Can_share(α,**x**,**y**,G_0)

- Can_share(α,**x**,**y**,G_0) iff there is an α edge from **x** to **y** in G_0 or if:
  - ∃ a vertex **s** ∈ G_0 with an **s** to **y** α edge,
  - ∃ a subject **x'** such that **x'**=**x** or **x'** initially spans to **x**,
  - ∃ a subject **s'** such that **s'**=**s** or **s'** terminally spans to **s**, and
  - ∃ islands $I_1, \ldots, I_n$ such that **x'** ∈ I_1, **s'** ∈ I_n, and there is a bridge from I_j to I_{j+1}
- Proof: If: **x'** grants to **x**, **s'** takes from **s**, otherwise as with subjects
  - Only if: as before, plus object can't give (receive) a right unless someone can take (grant) it
- Corollary: There is an $O(|V|+|E|)$ algorithm to test can_share

12

CS52600: Information Security
Protection Models

September 3, 2010

Prof. Chris Clifton

Creating models from scratch

- G_0 = ●, R a set of rights. $G_0 \vdash^* G$ iff G is a finite directed acyclic? graph, edges labeled from R, and at least one subject with no incoming edge.
  - If: construction (create)
  - Only if: Can't add an edge to initial subject
- A k-component, n-edge protection graph can be constructed from t-rule applications, where $2(k-1)+n \le t \le 2(k-1)+3n$

14

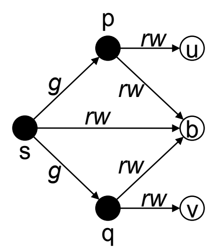
Use of the model

- Sharing rights with trusted entity
- Stealing (rights available with non-cooperating subjects)
- Collusion

15

Sharing Rights through Trusted Entity

- Subjects **p** and **q** communicate through buffer object **b**
  - Trusted entity **s** controls access to **b**
  - **p** and **q** have private information **u** and **v**



16

Theft

- Can_steal(α,**x**,**y**,G_0) if there is no α edge from **x** to **y** in G_0 and $\exists$ G_1, …, G_n s. t.:
  - $\exists$ α edge from **x** to **y** in $G_{n,}$,
  - $\exists$ rules $\rho_1,\ldots,\rho_n$ that take $G_{i-1} \vdash G_i$, and
  - $\forall$ **v**,**w** $\in G_i$, $1 \le i < n$, if $\exists$ α edge from **v** to **y** in G_0 then ρ_i is not "**v** grants (α to **y**) to **w**"
- Ideal: Steal possible if **x** gets α on **y** without anyone granting α on **y** to anyone

17

Theorem: When Theft Possible

- Can_steal(α,**x**,**y**,G_0) iff there is no α edge from **x** to **y** in G_0 and $\exists$ G_1, …, G_n s. t.:
  - $\exists$ subject **x'** such that **x'**=**x** or **x'** initially spans to **x**, and
  - $\exists$ **s** with α edge to **y** in G_0 and can_share(t,**x'**,**s**,G_0)
- Proof:
  - $\Rightarrow$: (easy – build path)
  - $\Leftarrow$: Assume can_steal:
    - No α edge from definition.
    - Can_share(α,**x**,**y**,G_0) from definition: α from **x** to **y** in G_n
    - **s** exists from can_share and Monday's theorem
    - Can_share(t,**x'**,**s**,G_0): **s** can't grant α (definition), someone else must get α from **s**, show that this can only be accomplished with take rule

18

Conspiracy

*How many subjects needed to enable Can_share(α,**x**,**y**,G_0)?*

- Access set A(**y**) with focus **y** is set of vertices **y** $\cup$ vertices to which **y** initially spans $\cup$ vertices to which **y** terminally spans
- Deletion set δ(**y**,**y'**): All **z** $\in$ A(**y**) $\cap$ A(**y'**) for which
  - **y** initially spans to **z** and **y'** terminally spans to **z** $\cup$
  - **y** terminally spans to **z** and **y'** initially spans to z $\cup$
  - **z=y** $\cup$ **z=y'**
- Conspiracy graph: if δ(**y**,**y'**) not empty, edge from **y** to **y'**

19

Conspiracy theorems:

- Can_share(α,**x**,**y**,G_0) iff conspiracy path from an item in an island containing **x** to an item that can steal from **y**
- Conspirators required is shortest above path in conspiracy graph

20

Protection Models: Do we have a contradiction?

- Harrison-Ruzzo-Ullman model (commands to change access control matrix
  - Safety undecidable
- Take-Grant Protection Model
  - Decidable in linear time
- What is the difference?
  - Restrictions on allowable operations
- What might we get with other sets of restrictions?

21

CS52600: Information Security
Protection Models

September 8, 2010
Prof. Chris Clifton

Schematic Protection Model

- Key idea: Protection Type τ
  - Label that determines how control rights affect an entity
  - Take-Grant: *subject* and *object* are different protection types
  - Unix file system: File, Directory, ???
- Ticket: Describes a set of rights
  - Entity has set *dom*(**X**) of tickets **Y**/*z* describing **X**'s rights *z* over entities **Y**
- Inert right vs. Control right
  - Inert right doesn't affect protection state

23

Transferring Rights

- Link predicate: $link_i(\mathbf{X},\mathbf{Y})$
  - conjunction or 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**
  - Determines if **X** and **Y** "connected" to transfer right
  - Example: $link(\mathbf{X},\mathbf{Y}) = \mathbf{Y}/g \in dom(\mathbf{X}) \vee \mathbf{X}/t \in dom(\mathbf{Y})$
- Filter function: conditions on transfer
- Copy $\mathbf{X}/r{:}c$ from **Y** to **Z** allowed iff $\exists i$ such that:
  - $\mathbf{X}/rc \in dom(\mathbf{Y})$
  - $link_i(\mathbf{Y},\mathbf{Z})$
  - $\tau(\mathbf{X})/r{:}c \in filter_i(\tau(\mathbf{Y}), \tau(\mathbf{Z}))$

24

Link Predicate

- Idea: $link_i(\mathbf{X}, \mathbf{Y})$ if **X** can assert some control right over **Y**
- Conjunction or 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**

25

Examples

- Take-Grant:

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

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

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

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

28

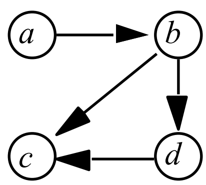
Example

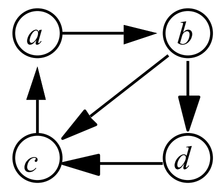
- Take-Grant Protection Model
  - TS = { subjects }, TO = { objects }
  - RC = { tc, gc }, RI = { rc, wc }
  - $link(\mathbf{p}, \mathbf{q}) = \mathbf{p}/t \in dom(\mathbf{q}) \vee \mathbf{q}/g \in dom(\mathbf{p})$
  - $f(subject, subject)$ = { $subject$, $object$ } $\times$ { tc, gc, rc, wc }
  - $f(subject,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

30

Create Operation

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





31

Types

- $cr(a, b)$: tickets introduced when subject of type *a* creates entity of type *b*
- **B** object: $cr(a, b) \subseteq \{ b/r{:}c \in RI \}$
- **B** subject: $cr(a, b)$ has two parts
  - $cr_P(a, b)$ 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 $cr_C(a, b)$

32

Non-Distinct Types

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

- *self*/*r*:*c* are tickets for creator
- *a*/*r*:*c* tickets for created

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

33

Attenuating Create Rule

cr(*a*, *b*) attenuating if:

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

34

Example: File Permissions

- Types: *users*, *files*
- (Inert) Rights: { *r:c*, *w:c*, *x:c* }
  - read, write, execute; copy on each
- $\forall$ **U**, **V** $\in$ users, *link*(**U**, **V**) = **true**
  - Anyone can grant a right to anyone if they posses the right to do so (copy)
- *f*(*user*, *user*) = { *file*/*r*, *file*/*w*, *file*/*x* }
  - Can copy read, write, execute
  - *But not copy right*

35

Safety Analysis in SPM

- Idea: derive *maximal state* where changes don't affect analysis
  - Similar to determining max flow
- Theorems:
  - A maximal state exists for every system
  - If parent gives child only rights parent has (conditions somewhat more complex), can easily derive maximal state

36

Typed Access Matrix Model

- Finite set T of types ($TS \subseteq T$ for subjects)
- Protection State: (S, O, τ, A)
  - τ:$O \rightarrow T$ is a type function
  - Operations same as Harrison-Ruzzo-Ullman except create adds type
- τ is child type iff command creates create subject/object of type τ (otherwise parent)
- If parent/child graph from all commands acyclic, then:
  - Safety is decidable
  - Safety is NP-Hard
  - Safety is polynomial if all commands limited to three parameters

37

Comparing Models

- Expressive Power
  - HRU/Access Control Matrix subsumes Take-Grant
  - HRU subsumes Typed Access Control Matrix
  - SPM subsumes Take-Grant
    - Subject/Object protection types
    - ticket is label on an edge
    - take/grant are control rights
- What about SPM and HRU?
  - SPM has no revocation (delete/destroy)
- HRU without delete/destroy (monotonic HRU)?
  - MTAM subsumes monotonic mono-operational HRU
  - HRU can have create requiring multiple “parents”

38

Extended Schematic Protection Model

- Adds "joint create": new node has multiple parents
  - Allows more natural representation of sharing between mutually suspicious parties
    - Create joint node for sharing
  - In Take-Grant, SPM, must create two nodes, they interact to share (equivalent power)
- Monotonic ESPM and Monotonic HRU equivalent

40

Multiparent Create

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

```
command multicreate(s0, s1, o)
if r in a[s0, s1] and r in a[s1, s0]
then
  create object o;
  enter r into a[s0, o];
  enter r into a[s1, o];
end
```

41

SPM and Multiparent Create

- can•create 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
  - $cr_{P,i}(\tau(\mathbf{X}_1), \ldots, \tau(\mathbf{X}_n)) = \mathbf{Y}/R_{1,1} \cup \mathbf{X}_i/R_{2,i}$
  - $cr_C(\tau(\mathbf{X}_1), \ldots, \tau(\mathbf{X}_n)) = \mathbf{Y}/R_3 \cup \mathbf{X}_1/R_{4,1} \cup \ldots \cup \mathbf{X}_n/R_{4,n}$

42

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 $x \in R$
  - $cc(a, a) = p$
  - $cr_{\text{Anna}}(a, a, p) = cr_{\text{Bill}}(a, a, p) = \varnothing$
  - $cr_{\text{proxy}}(a, a, p) = \{ \text{Anna}/x, \text{Bill}/x \}$

43

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 $\mathbf{C}$):
  - $cc(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = Z \subseteq T$
  - $cr_{\mathbf{P}1}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{1,1} \cup \mathbf{P}_1/R_{2,1}$
  - $cr_{\mathbf{P}2}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{2,1} \cup \mathbf{P}_2/R_{2,2}$
  - $cr_{\mathbf{P}3}(\tau(\mathbf{P}_1), \tau(\mathbf{P}_2), \tau(\mathbf{P}_3)) = \mathbf{C}/R_{3,1} \cup \mathbf{P}_3/R_{2,3}$

44

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)

45

Entities and Types

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

46

Can•Create

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

47

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)

48

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

49

Link Predicates

- Idea: no tickets to parents until child created
  - Done by requiring each agent to have its own parent rights
  - $link_1(\mathbf{A}_1, \mathbf{A}_2) = \mathbf{A}_1/t \in dom(\mathbf{A}_2)$ ② $\mathbf{A}_2/t \in dom(\mathbf{A}_2)$
  - $link_1(\mathbf{A}_2, \mathbf{A}_3) = \mathbf{A}_2/t \in dom(\mathbf{A}_3)$ ② $\mathbf{A}_3/t \in dom(\mathbf{A}_3)$
  - $link_2(\mathbf{S}, \mathbf{A}_3) = \mathbf{A}_3/t \in dom(\mathbf{S})$ ② $\mathbf{C}/t \in dom(\mathbf{C})$
  - $link_3(\mathbf{A}_1, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_1)$
  - $link_3(\mathbf{A}_2, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_2)$
  - $link_3(\mathbf{A}_3, \mathbf{C}) = \mathbf{C}/t \in dom(\mathbf{A}_3)$
  - $link_4(\mathbf{A}_1, \mathbf{P}_1) = \mathbf{P}_1/t \in dom(\mathbf{A}_1)$ ② $\mathbf{A}_1/t \in dom(\mathbf{A}_1)$
  - $link_4(\mathbf{A}_2, \mathbf{P}_2) = \mathbf{P}_2/t \in dom(\mathbf{A}_2)$ ② $\mathbf{A}_2/t \in dom(\mathbf{A}_2)$
  - $link_4(\mathbf{A}_3, \mathbf{P}_3) = \mathbf{P}_3/t \in dom(\mathbf{A}_3)$ ② $\mathbf{A}_3/t \in dom(\mathbf{A}_3)$

50

Filter Functions

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

51

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 $\mathbf{P}_2/Rtc$ u $\mathbf{A}_1/tc$
- $\mathbf{A}_3$ has $\mathbf{P}_3/Rtc$ u $\mathbf{A}_2/tc$
- **S** has $\mathbf{A}_3/tc$ u $\mathbf{C}/Rtc$
- **C** has $\mathbf{C}/R_3$

52

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 $\mathbf{P}_2/Rtc \cup \mathbf{A}_1/tc \cup \mathbf{A}_1/t \cup \mathbf{C}/Rtc$
- Now all $link_3$s true $\Rightarrow$ apply f_3
  - **C** has $\mathbf{C}/R_3 \cup \mathbf{P}_1/R_{4,1} \cup \mathbf{P}_2/R_{4,2} \cup \mathbf{P}_3/R_{4,3}$

53

Finish Construction

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

54

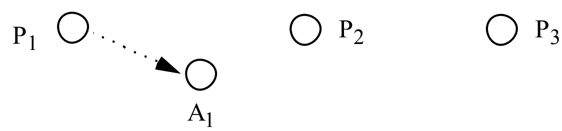
Theorem

- 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

55

Example: 3-Parent Joint Creation

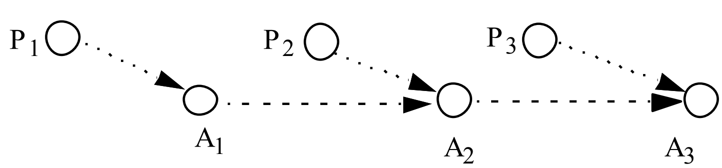
- Simulate with 2-parent
  - Nodes **P**$_1$, **P**$_2$, **P**$_3$ parents
  - Create node **C** with type *c* with edges of type *e*
  - Add node **A**$_1$ of type *a* and edge from **P**$_1$ to **A**$_1$ of type *e´*



56

Next Step

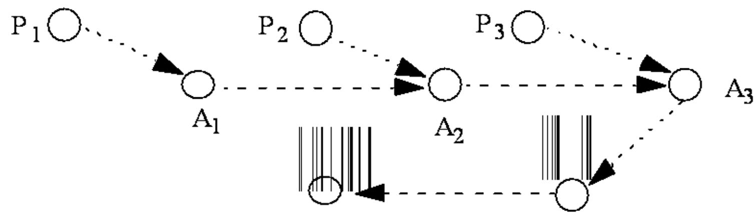
- **A**$_1$, **P**$_2$ create **A**$_2$; **A**$_2$, **P**$_3$ create **A**$_3$
- Type of nodes, edges are *a* and *e´*



57

Next Step

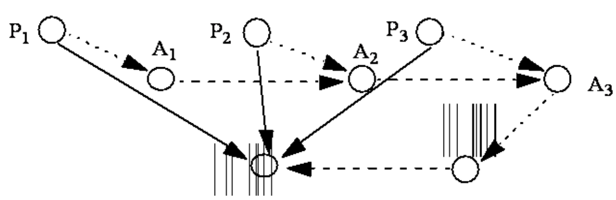
- $\mathbf{A_3}$ creates **S**, of type *a*
- **S** creates **C**, of type *c*



58

Last Step

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



59

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

60

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*

61

Theorems

- Monotonic ESPM and the monotonic HRU model are equivalent.
- Safety question in ESPM also decidable if acyclic attenuating scheme

63

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

64

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*

65

Expressive Power

- If scheme in *MA* 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

66

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?

67

Can *A* Simulate *B*?

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

68

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

69

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*

70

Theorem

- Monotonic single-parent models are less expressive than monotonic multiparent models
- ESPM more expressive than SPM
  - ESPM multiparent and monotonic
  - SPM monotonic but single parent

71

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*), where τ:*O*$\rightarrow$*T* specifies type of each object
  - If **X** subject, τ(**X**) in *TS*
  - If **X** object, τ(**X**) in *T* – *TS*

72

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*

73

Create Subject

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

74

Create Object

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

75

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

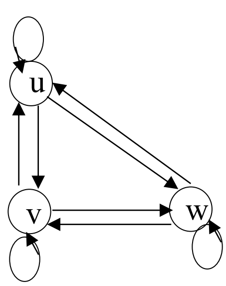
76

Cyclic Creates

command *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

77

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

78

Theorems

- Safety decidable for systems with acyclic MTAM schemes
- 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

79

Key Points

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

80