

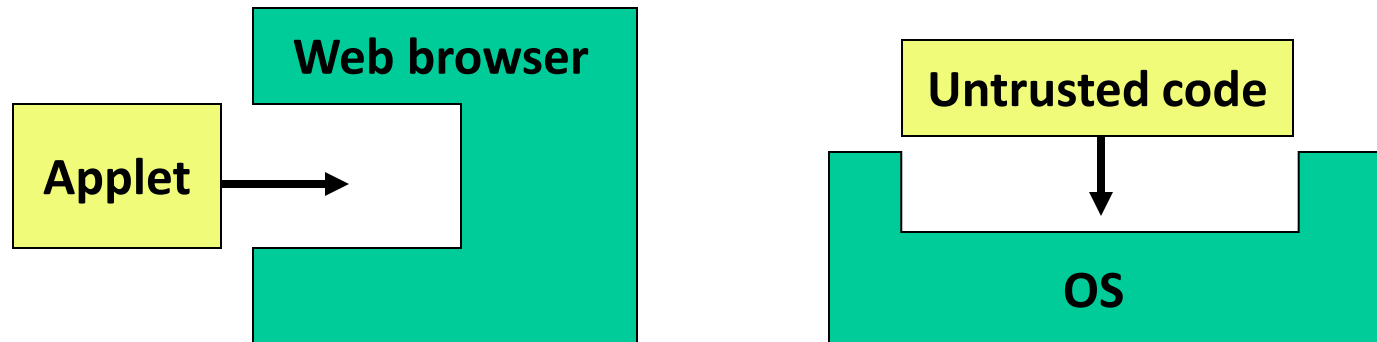
# Run-time Program Monitors: **Theory** and Practice

Lujo Bauer

18-732

Fall 2010

# Securing Extensible Systems



- What security policies can we enforce?
- What mechanisms can we use?

# Enforcement in Software

- Static analyses
  - Type checking, bytecode verification
- Run-time enforcement
  - Java stack inspection
- Program rewriting

# An Ideal Enforcement Mechanism

- General
  - Many policies
- Easy to reason about
  - A sound theory
- Simple to apply
  - A convenient specification language

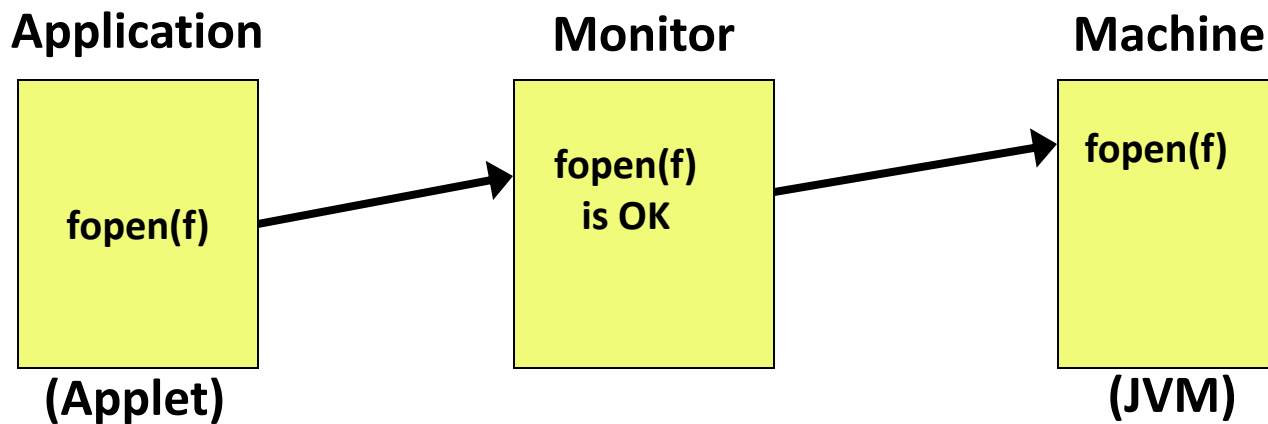
# Enforcement in Software

- Static analyses
  - Type checking, bytecode verification
- Run-time enforcement
  - Java stack inspection
- Program rewriting

# Run-time Enforcement: Program Monitors

- A program monitor is a coroutine that runs in parallel with an untrusted application
- Monitors process security-relevant actions
  - decide to allow/disallow application actions
  - disallow by terminating application

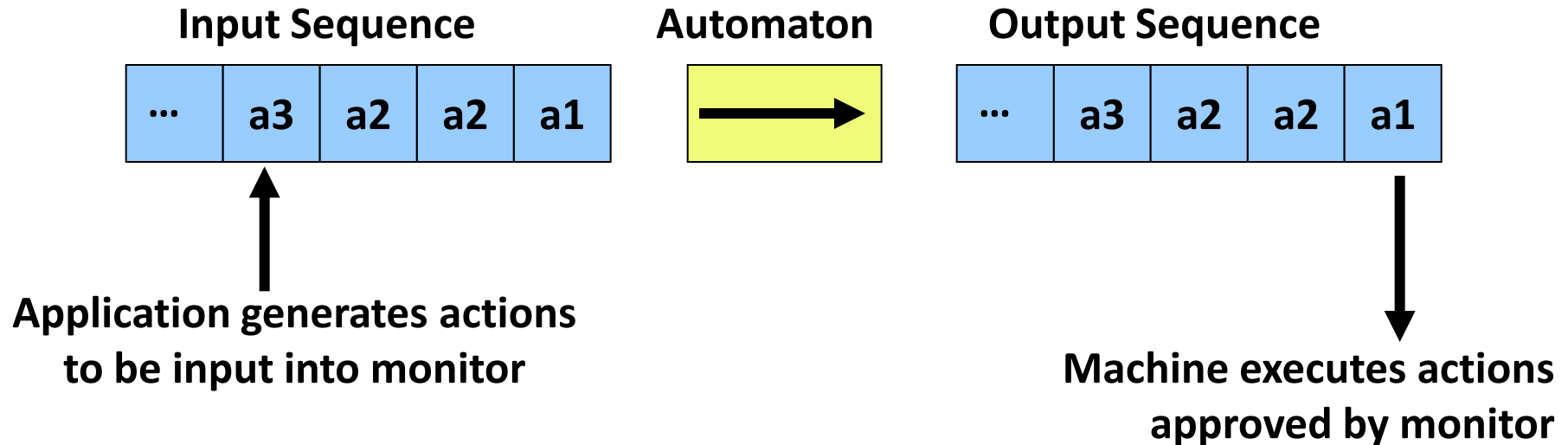
# Run-time Enforcement: Program Monitors



# Program Monitors: Theory & Practice

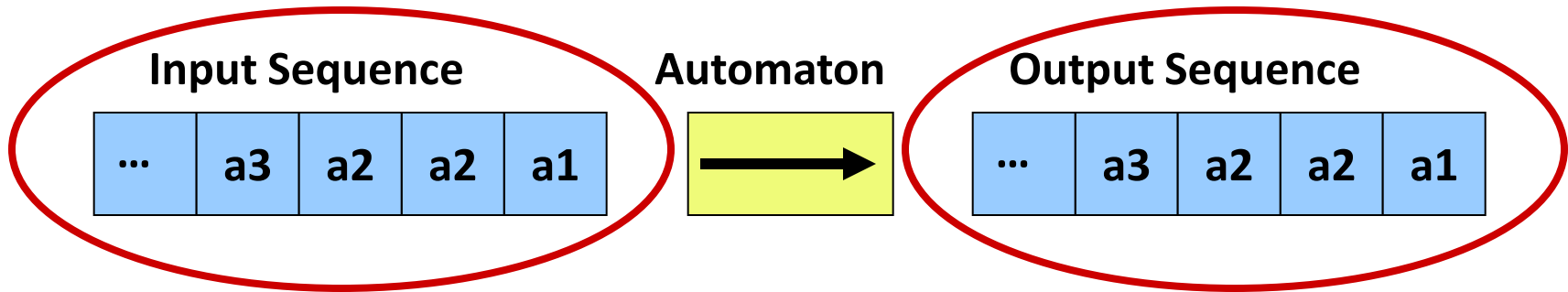
- Theory
  - What is a program monitor?
  - What is a policy and what does it mean to enforce it?
  - What policies can monitors enforce?
- Practice
  - How do we implement program monitors?
  - How do we specify policies?

# An Abstract Model: Security Automata



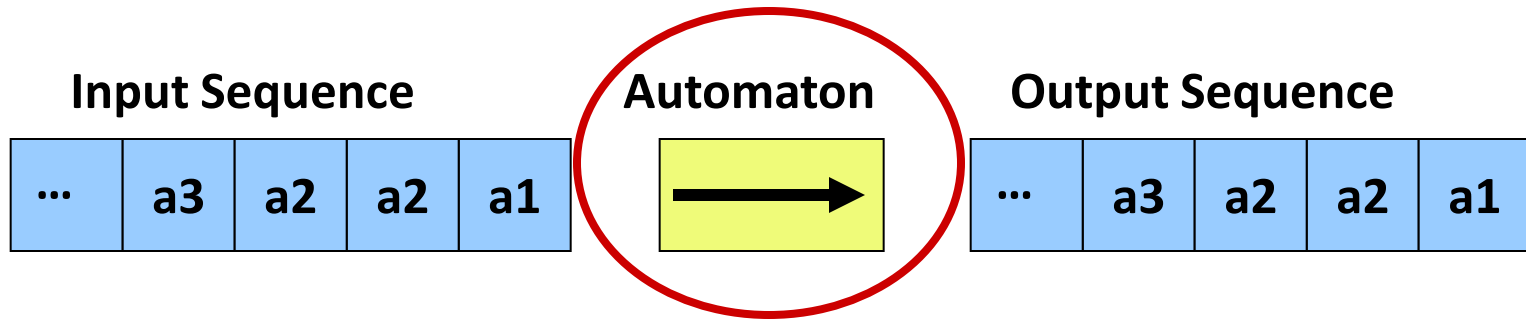
- Formal models of program monitors
- Observe/monitor a sequence of untrusted actions

# Security Automata



- $\sigma$  : an execution (a sequence of actions)
- $\Psi$  : universe of all possible sequences
  - Finite or infinite
- $\Sigma_S$  : subset of  $\Psi$  corresponding to the executions of target  $S$

# Security Automata



- Deterministic finite- or infinite-state machine  
 $(Q, Q_0, \delta)$

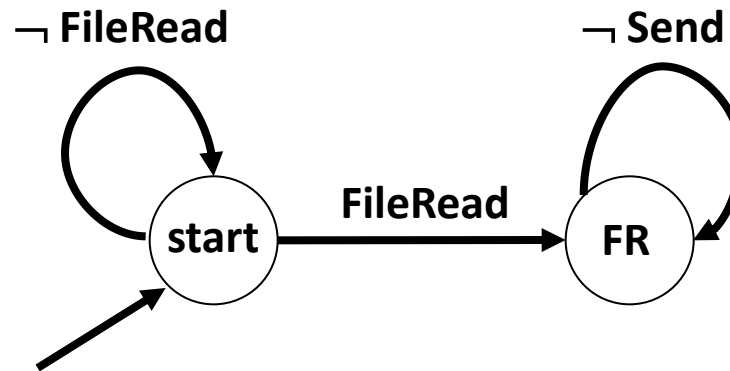
$Q$  = state set

$Q_0 \subseteq Q$  = set of start states

$\delta : (Q \times I) \rightarrow 2^Q$  = transition function

$I$  = set of input symbols

# Security Automata: Example



- “No send after file read” policy
- $I = \{ \text{FileRead}, \text{Send}, \dots \}$
- $Q_0 = \{ \text{start} \}$

# Program Monitors: Theory & Practice

- Theory
  - What is a program monitor?
  - **What is a policy and what does it mean to enforce it?**
  - What policies can monitors enforce?
- Practice
  - How do we implement program monitors?
  - How do we specify policies?

# Policies

- “A security policy is specified by giving a predicate on sets of executions. A target  $S$  satisfies security policy  $\mathcal{P}$  if and only if  $\mathcal{P}(\Sigma_S)$  equals true.” [Schneider]

# Policies and Properties

- Policies:

$$\mathcal{P}(\Sigma)$$

- Properties:

$$\mathcal{P}(\Sigma) : (\forall \sigma \in \Sigma : \hat{\mathcal{P}}(\sigma))$$

- Policies hold on sets
- Policies are properties if they can be defined by predicates that hold on individual executions

# Policies and Properties

- Policies
  - Cryptographic uniformity
  - Nonce uniqueness
- Properties
  - Access control

# Enforceable Properties

- Safety properties  $\Gamma \subseteq \Sigma$   
 $\sigma \notin \Gamma \Rightarrow \exists i: (\forall \tau \in \Psi: \sigma[..i]\tau \notin \Gamma)$

# Enforceable Properties

- Safety properties  $\Gamma \subseteq \Sigma$   
$$\sigma \notin \Gamma \Rightarrow \exists i: (\forall \tau \in \Psi: \sigma[..i]\tau \notin \Gamma)$$
- No “bad thing” will happen
- “Bad things” cannot be undone
- Schneider’s security automata enforce exactly the set of safety properties

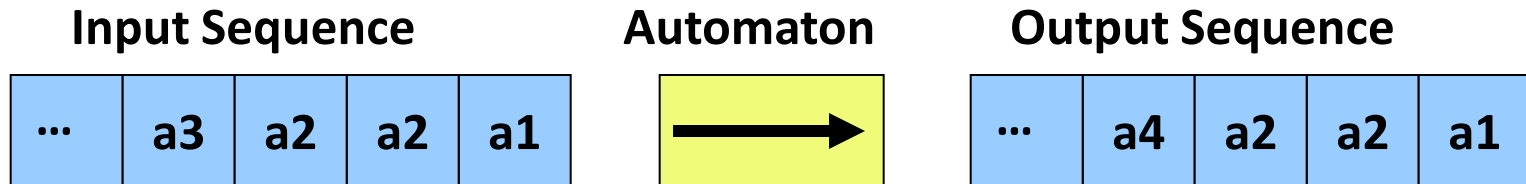
# Program Monitors: Theory & Practice

- Theory
  - What is a program monitor?
  - What is a policy and what does it mean to enforce it?
  - **What policies can monitors enforce?**
- Practice
  - How do we implement program monitors?
  - How do we specify policies?

# Enforcing More Properties

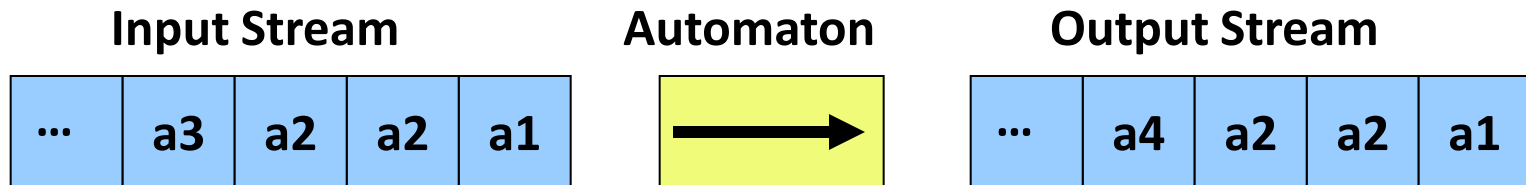
- Give monitors more power
  - Halt, OK, ...
- Give monitors more latitude
  - Enforce precisely, conservatively, ...

# More Powerful Security Automata



- **Accept** the action
  - **Halt** the application
  - **Suppress** (skip) the operation
  - **Insert** some computation
- 
- Monitors are sequence **transformers**

# More Powerful Security Automata



- Single step (determined by  $\delta$ ):

$$(\sigma_{in}, q) \xrightarrow{\sigma_{out}} (\sigma_{in}', q')$$

- Output sequence is observable
- Input sequences are not observable

# Types of Automata

	Insert	Suppress	OK	Halt
Truncation			✓	✓
Suppression		✓	✓	✓
Insertion	✓		✓	✓
Edit	✓	✓	✓	✓

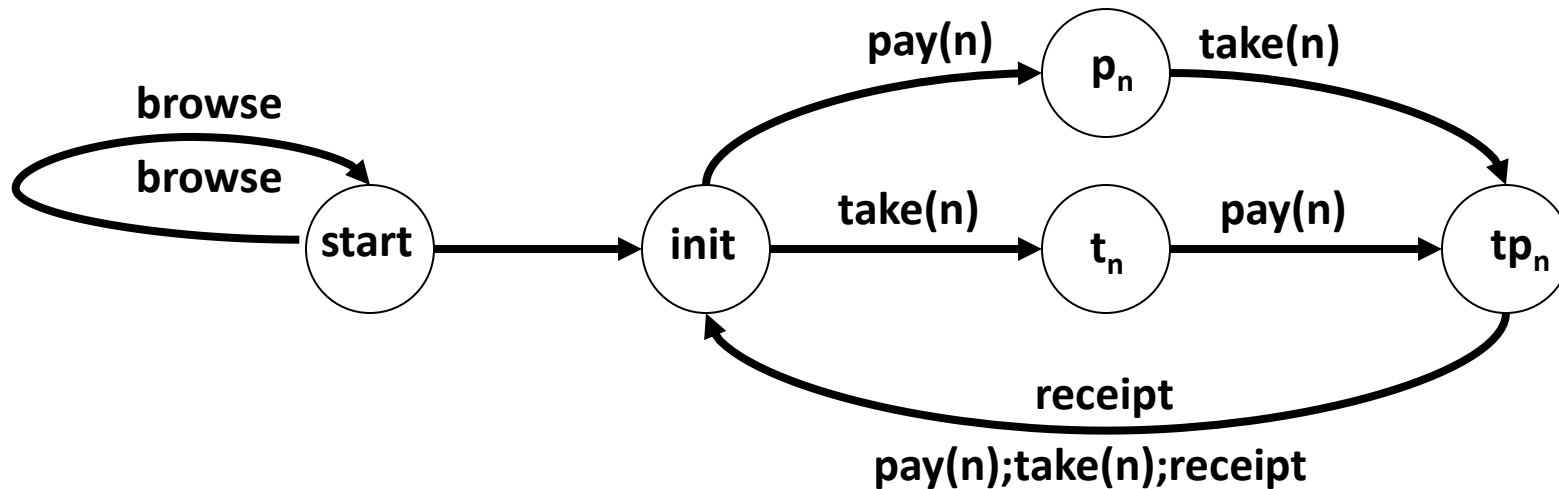
# Edit Automata: The Banana Market

- Set of application actions:

```

A = { take(n),           // take n bananas
      pay(n),            // pay for n bananas
      browse,           // browse for bananas
      receipt           // commit
    }
  
```

- Edit Automaton:



# With Power Comes Responsibility...

- Is it OK to suppress?
  - Is it necessary?
  - Is it allowed?
  - Could it be helpful?
- Is it OK to insert?



# Principles of Enforcement

- Principle of **Soundness**

All observable outputs obey the policy

$\forall$  sequences  $\sigma_{in}$ .  $\exists$  state  $q'$ .  $\exists$  sequence  $\sigma_{out}$

1.  $(\sigma_{in}, q_0) \Rightarrow (\text{empty}, q')$
2.  $P(\sigma_{out})$

- Principle of **Transparency**

Semantics of executions that already obey policy must be preserved

3.  $P(\sigma_{in}) \Rightarrow (\sigma_{in} \cong \sigma_{out})$

# Semantic Equivalence ( $\cong$ )

- Necessary properties
  - Reflexive, symmetric, transitive
  - $\sigma \cong \sigma' \Rightarrow P(\sigma) \Leftrightarrow P(\sigma')$
- Examples
  - $\text{fclose}(f); \text{fclose}(f) \cong \text{fclose}(f)$
  - $\text{fopen}(f); \text{fopen}(g) \cong \text{fopen}(g); \text{fopen}(f)$

# Conservative Enforcement

- Automaton satisfies Soundness but not necessarily Transparency
- $\forall$  properties  $P$  .  
     $(\exists \text{ sequence } \sigma . P(\sigma)) \Rightarrow$   
     $P$  can be conservatively enforced

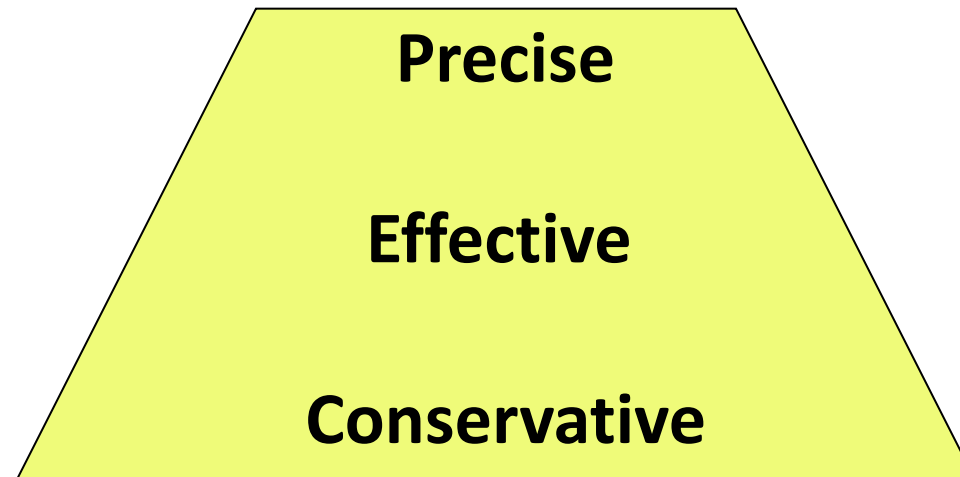
# Precise Enforcement

- Automaton satisfies Soundness and Transparency
- Valid sequences cannot be altered in any way, at any time

# Effective Enforcement

- Automaton satisfies Soundness and Transparency
- Valid sequences can be altered (with certain conditions)

# Enforcement Types



# Edit Automata and Effective Enforcement

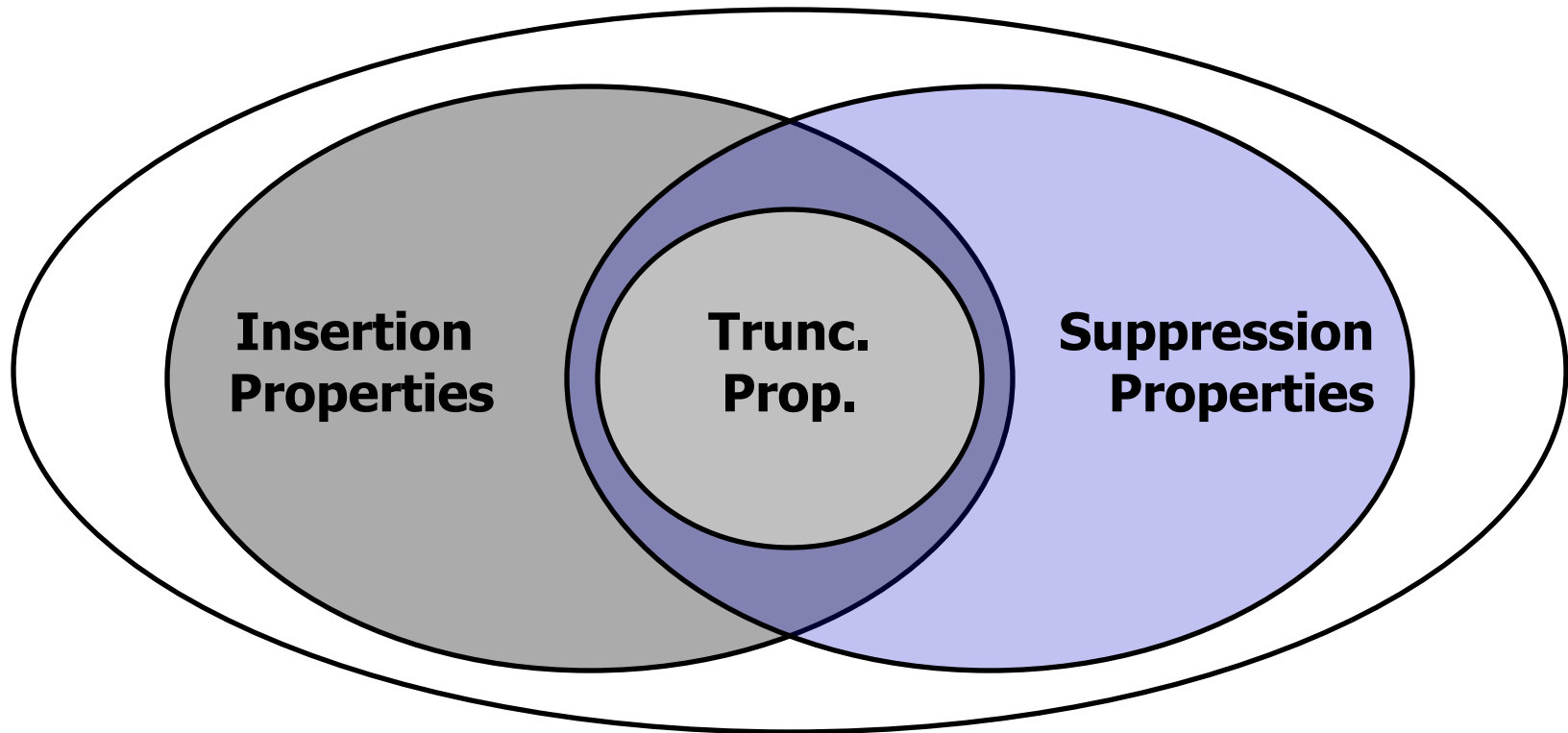
- A banana market policy
  - $\text{browse}^*; ((\text{take}(n);\text{pay}(n) \mid \text{pay}(n);\text{take}(n)) ; \text{receipt})^*$
  - $\text{take}(n);\text{pay}(n) \cong \text{pay}(n);\text{take}(n)$
  - Our edit automaton is an effective enforcer:
    - It satisfies Soundness
    - It satisfies Transparency
    - Proofs are by induction over the possible inputs
  - Less powerful automata (truncation, suppression and insertion) cannot enforce the banana market property
    - Proof by contradiction shows either Soundness or Transparency will be violated

# What Can Be Enforced?

- The enforceable properties depend upon
  - the **definition of enforcement** (conservative, effective, precise)
  - the **class of automaton** (truncation, suppression, insertion, edit)
  - the **space of possible input programs**
    - if the monitor can assume certain “bad” executions do not occur, it can enforce more properties
    - static program analysis (type systems; proof-carrying code) can constrain program execution in ways useful to run-time monitors

# Effective Enforceable Properties

## Editing Properties



# Program Monitors: Theory & Practice

- Theory
  - What is a program monitor?
  - What is a policy and what does it mean to enforce it?
  - What policies can monitors enforce?
- Practice
  - How do we implement program monitors?
  - How do we specify policies?

# Sources

- F. Schneider. Enforceable Security Properties. *ACM Transactions on Information and System Security*, 3(1), 30–50, 2000.
- J. Ligatti, L. Bauer, and D. Walker. Enforcing non-safety security policies with program monitors. In *Computer Security—ESORICS 2005: 10th European Symposium on Research in Computer Security, Lecture Notes in Computer Science*, 3679, 355–373, 2005.