

# A rewrite-based approach for security policies

Hélène Kirchner<sup>1</sup>

Joint work with  
Claude Kirchner<sup>1</sup>, Anderson Santana de Oliveira<sup>2</sup>,  
Dan Dougherty<sup>3</sup> and Eric Ke Wang<sup>4</sup>

INRIA Bordeaux - Sud-Ouest Research Center, France

Universidade Federal do Rio Grande do Norte, Brasil

Worcester Polytechnic Institute, USA

University of Hong Kong, China

November 2008

Security policies are required to organize access to resources and protect the them from being processed by undesirable users (subjects).

*Ex: Access control policies expressed with authorization rules.*

*How to design highly dependable security policies that ensure secure access to distributed resources ?*

In large systems, different subjects usually have different (even competing) requirements on the use of resources and their security goals may be distinct. Hence, various access requirements have to be consistently authorized and maintained in a single policy.

**Formal methods for the design and verification of security policies** (Pareo team)

Apply a **rewrite-based approach** to the compositional design, maintenance and verification of security policies.

- 1 **Rewrite rules** transform input terms representing policy requests into decision terms.  
**Strategies** are used to explicitly control the rule application.
- 2 Analysis is performed through parameterized queries solved by **narrowing**.
- 3 Existing proof techniques and analysis tools can be reused in the security policies context.

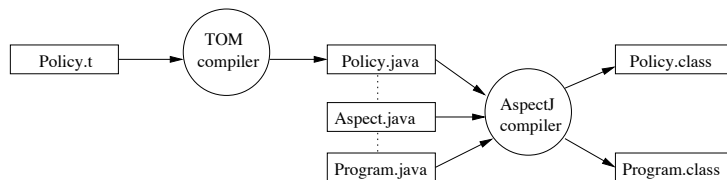
- Policy properties:
  - *Does a policy always return a decision?* **termination, completeness**
  - *Is a policy consistent (one decision for each request)?* **confluence**
  - *Does policy composition preserve the properties of the component policies?* **modularity results**

[Dougherty&all-Esorics07],[SantanaDeOliveiraPhD2008]

- Requests analysis: *what if a request is made under these conditions?* **narrowing provides abstraction of the rewriting derivation trees/request evaluation.** [Kirchner&all-Secret08]

# Design tool and implementation

A methodology to weave rewrite-based policies into existing Java programs ([FMSE07]) using Tom (<http://tom.loria.fr>) that extends Java with pattern matching, rewriting, and strategies



- 1 Rewrite-Based Policies
- 2 Properties of policies
- 3 Policy Composition
- 4 Policy Analysis
- 5 Policy enforcement and implementation
- 6 Conclusion

# Example - A firewall policy

*Block any traffic coming from the outside of the local network.*

A packet has a source, a destination and a state.

<i>pkt</i>	:	<i>Address</i> × <i>Address</i> × <i>State</i>	↦	<i>Decision</i>
<i>new, est</i>	:		↦	<i>State</i>
<i>drop, accept</i>	:		↦	<i>Decision</i>
<i>eth0, ppp0, 10.1.1.1,</i> <i>10.1.1.2, 123.123.1.1</i>	:		↦	<i>Address</i>

# Example - Firewall rules

Packets concerning established connections have to be accepted.

$$pkt(src, dst, \mathbf{est}) \rightarrow \mathit{accept}$$

Any new packet whose origin is the local network is accepted.

$$pkt(\mathbf{eth0}, dst, \mathbf{new}) \rightarrow \mathit{accept}$$

Any new packet coming from the external network (*ppp0*) is rejected.

$$pkt(\mathbf{ppp0}, dst, \mathbf{new}) \rightarrow \mathit{drop}$$

Some IP address of the local machines are rewritten to a single IP address visible from the external network.

$$pkt(\mathbf{10.1.1.1}, \mathbf{ppp0}, s) \rightarrow pkt(\mathbf{123.123.1.1}, \mathbf{ppp0}, s)$$
$$pkt(\mathbf{10.1.1.2}, \mathbf{ppp0}, s) \rightarrow pkt(\mathbf{123.123.1.1}, \mathbf{ppp0}, s)$$

# Example - Firewall rules

Packets concerning established connections have to be accepted.

$$pkt(src, dst, \mathbf{est}) \rightarrow \mathit{accept}$$

Any new packet whose origin is the local network is accepted.

$$pkt(\mathbf{eth0}, dst, \mathbf{new}) \rightarrow \mathit{accept}$$

Any new packet coming from the external network (*ppp0*) is rejected.

$$pkt(\mathbf{ppp0}, dst, \mathbf{new}) \rightarrow \mathit{drop}$$

Some IP address of the local machines are rewritten to a single IP address visible from the external network.

$$pkt(10.1.1.1, \mathbf{ppp0}, s) \rightarrow pkt(123.123.1.1, \mathbf{ppp0}, s)$$
$$pkt(10.1.1.2, \mathbf{ppp0}, s) \rightarrow pkt(123.123.1.1, \mathbf{ppp0}, s)$$

# Example - Firewall rules

Packets concerning established connections have to be accepted.

$$pkt(src, dst, \mathbf{est}) \rightarrow \mathit{accept}$$

Any new packet whose origin is the local network is accepted.

$$pkt(\mathbf{eth0}, dst, \mathbf{new}) \rightarrow \mathit{accept}$$

Any new packet coming from the external network (*ppp0*) is rejected.

$$pkt(\mathbf{ppp0}, dst, \mathbf{new}) \rightarrow \mathit{drop}$$

Some IP address of the local machines are rewritten to a single IP address visible from the external network.

$$pkt(10.1.1.1, \mathbf{ppp0}, s) \rightarrow pkt(123.123.1.1, \mathbf{ppp0}, s)$$
$$pkt(10.1.1.2, \mathbf{ppp0}, s) \rightarrow pkt(123.123.1.1, \mathbf{ppp0}, s)$$

# Example - Firewall rules

Packets concerning established connections have to be accepted.

$$pkt(src, dst, \mathbf{est}) \rightarrow \mathit{accept}$$

Any new packet whose origin is the local network is accepted.

$$pkt(\mathbf{eth0}, dst, \mathbf{new}) \rightarrow \mathit{accept}$$

Any new packet coming from the external network (*ppp0*) is rejected.

$$pkt(\mathbf{ppp0}, dst, \mathbf{new}) \rightarrow \mathit{drop}$$

Some IP address of the local machines are rewritten to a single IP address visible from the external network.

$$pkt(\mathbf{10.1.1.1}, \mathbf{ppp0}, s) \rightarrow pkt(\mathbf{123.123.1.1}, \mathbf{ppp0}, s)$$
$$pkt(\mathbf{10.1.1.2}, \mathbf{ppp0}, s) \rightarrow pkt(\mathbf{123.123.1.1}, \mathbf{ppp0}, s)$$

A security policy  $\wp$  is defined as  $(\mathcal{F}, D, R, Q, \zeta)$  where :

- 1  $\mathcal{F}$  is a signature – *for defining the data structure for the policy environment*
- 2  $D$  is a non-empty set of ground terms – *which represent decisions*
- 3  $R$  is a set of rewrite rules – *to formalize the rules of the policy*
- 4  $Q$  a set of ground terms – *that represent requests*
- 5  $\zeta$  is a rewrite strategy – *that controls rule application.*

A security policy  $\wp$  is defined as  $(\mathcal{F}, D, R, Q, \zeta)$  where :

- 1  $\mathcal{F}$  is a signature – *for defining the data structure for the policy environment*
- 2  $D$  is a non-empty set of ground terms – *which represent decisions*
- 3  $R$  is a set of rewrite rules – *to formalize the rules of the policy*
- 4  $Q$  a set of ground terms – *that represent requests*
- 5  $\zeta$  is a rewrite strategy – *that controls rule application.*

A security policy  $\wp$  is defined as  $(\mathcal{F}, D, R, Q, \zeta)$  where :

- 1  $\mathcal{F}$  is a signature – *for defining the data structure for the policy environment*
- 2  $D$  is a non-empty set of ground terms – *which represent decisions*
- 3  $R$  is a set of rewrite rules – *to formalize the rules of the policy*
- 4  $Q$  a set of ground terms – *that represent requests*
- 5  $\zeta$  is a rewrite strategy – *that controls rule application.*

A security policy  $\wp$  is defined as  $(\mathcal{F}, D, R, Q, \zeta)$  where :

- 1  $\mathcal{F}$  is a signature – *for defining the data structure for the policy environment*
- 2  $D$  is a non-empty set of ground terms – *which represent decisions*
- 3  $R$  is a set of rewrite rules – *to formalize the rules of the policy*
- 4  $Q$  a set of ground terms – *that represent requests*
- 5  $\zeta$  is a rewrite strategy – *that controls rule application.*

A security policy  $\wp$  is defined as  $(\mathcal{F}, D, R, Q, \zeta)$  where :

- 1  $\mathcal{F}$  is a signature – *for defining the data structure for the policy environment*
- 2  $D$  is a non-empty set of ground terms – *which represent decisions*
- 3  $R$  is a set of rewrite rules – *to formalize the rules of the policy*
- 4  $Q$  a set of ground terms – *that represent requests*
- 5  $\zeta$  is a rewrite strategy – *that controls rule application.*

A security policy  $\wp$  is defined as  $(\mathcal{F}, D, R, Q, \zeta)$  where :

- 1  $\mathcal{F}$  is a signature – *for defining the data structure for the policy environment*
- 2  $D$  is a non-empty set of ground terms – *which represent decisions*
- 3  $R$  is a set of rewrite rules – *to formalize the rules of the policy*
- 4  $Q$  a set of ground terms – *that represent requests*
- 5  $\zeta$  is a rewrite strategy – *that controls rule application.*

# Example - A firewall policy

Let the policy sorted signature  $\mathcal{F}$  be:

$pkt$	:	$Address \times Address \times State$	$\mapsto$	$Decision$
$new, est$	:		$\mapsto$	$State$
$drop, accept$	:		$\mapsto$	$Decision$
$eth0, ppp0, 10.1.1.1,$ $10.1.1.2, 123.123.1.1$	:		$\mapsto$	$Address$

$D = \{accept, drop\}$ .

$R$  is the set of rewrite rules

$pkt(src, dst, est)$	$\rightarrow$	$accept$
$pkt(eth0, dst, new)$	$\rightarrow$	$accept$
$pkt(ppp0, dst, new)$	$\rightarrow$	$drop$
$pkt(10.1.1.1, ppp0, s)$	$\rightarrow$	$pkt(123.123.1.1, ppp0, s)$
$pkt(10.1.1.2, ppp0, s)$	$\rightarrow$	$pkt(123.123.1.1, ppp0, s)$

The set  $Q$  consists in all ground terms having the symbol  $pkt$  at top position.

A common strategy for such firewall policy, is to apply rules in the order they are given.

# Example - A conference policy

*Authors can submit papers during the submission phase.*

*aut(q(author(x), submitPaper, paper(x, z)), submission, cnd)* → permit

*aut(q(author(x), readScores, paper(x, z)), phase, cnd)* → deny

*aut(q(reviewee(x), action, p), phase, conflict(x, p))* → deny

*aut(q(reviewee(x), action, paper(x, z)), phase, cnd)* →

*aut(q(reviewee(x), action, paper(x, z)), phase, conflict(x, paper(x, z)))*

*aut(a, b, c)* → notApp

# Example - A conference policy

*Authors can submit papers during the submission phase.*

*aut(q(author(x), **submitPaper**, paper(x, z)), **submission**, *cond*)* → **permit**

*aut(q(author(x), **readScores**, paper(x, z)), *phase*, *cond*)* → **deny**

*aut(q(reviewer(x), *action*, p), *phase*, conflict(x, p))* → **deny**

*aut(q(reviewer(x), *action*, paper(x, z)), *phase*, *cond*)* →

*aut(q(reviewer(x), *action*, paper(x, z)), *phase*, conflict(x, paper(x, z)))*

*aut(a, b, c)* → **notApp**

# Example - A conference policy

*Authors may never read scores for his paper.*

$aut(q(author(x), submitPaper, paper(x, z)), submission, cnd) \rightarrow permit$

$aut(q(author(x), readScores, paper(x, z)), phase, cnd) \rightarrow deny$

$aut(q(reviewer(x), action, p), phase, conflict(x, p)) \rightarrow deny$

$aut(q(reviewer(x), action, paper(x, z)), phase, cnd) \rightarrow$

$aut(q(reviewer(x), action, paper(x, z)), phase, conflict(x, paper(x, z)))$

$aut(a, b, c) \rightarrow notApp$

# Example - A conference policy

*A reviewer may never perform an action over a paper if he is conflicted with that paper.*

*aut(q(author(x), **submitPaper**, paper(x, z)), **submission**, *cond*)* → **permit**

*aut(q(author(x), **readScores**, paper(x, z)), *phase*, *cond*)* → **deny**

*aut(q(reviewers(x), *action*, p), *phase*, conflict(x, p))* → **deny**

*aut(q(reviewers(x), *action*, paper(x, z)), *phase*, *cond*)* →

*aut(q(reviewers(x), *action*, paper(x, z)), *phase*, conflict(x, paper(x, z)))*

*aut(a, b, c)* → **notApp**

# Example - A conference policy

*A reviewer is conflicted with a paper if he is an author for that paper.*

$aut(q(author(x), submitPaper, paper(x, z)), submission, cnd) \rightarrow permit$

$aut(q(author(x), readScores, paper(x, z)), phase, cnd) \rightarrow deny$

$aut(q(reviewer(x), action, p), phase, conflict(x, p)) \rightarrow deny$

$aut(q(reviewer(x), action, paper(x, z)), phase, cnd) \rightarrow$

$aut(q(reviewer(x), action, paper(x, z)), phase, conflict(x, paper(x, z)))$

$aut(a, b, c) \rightarrow notApp$

# Example - A conference policy

*In any other case, the policy is not applicable*

$aut(q(author(x), submitPaper, paper(x, z)), submission, cnd) \rightarrow permit$

$aut(q(author(x), readScores, paper(x, z)), phase, cnd) \rightarrow deny$

$aut(q(reviewer(x), action, p), phase, conflict(x, p)) \rightarrow deny$

$aut(q(reviewer(x), action, paper(x, z)), phase, cnd) \rightarrow$

$aut(q(reviewer(x), action, paper(x, z)), phase, conflict(x, paper(x, z)))$

$aut(a, b, c) \rightarrow notApp$

# Rewriting and Narrowing

Inside the machinery

Given a rewrite system  $R$ ,

$$t \rightarrow_{\omega, l \rightarrow r, \sigma}^R t'$$

if there exist  $l \rightarrow r$ ,  $\omega$  in  $t$ , and  $\sigma$  such that  $t|_{\omega} = \sigma l$ .

Then  $t' = t[\sigma r]_{\omega}$ .

$$t \rightsquigarrow_{\omega, l \rightarrow r, \sigma}^R t'$$

if there exist  $l \rightarrow r$ ,  $\omega$  in  $t$  and  $\sigma$  such that  $\sigma t|_{\omega} = \sigma l$ .

Then  $t' = \sigma(t[r]_{\omega})$ .

Restricted notions of Strategic Rewriting and Narrowing.

[Dougherty&all-Esorics07],[SantanaDeOliveiraPhD2008]

A security policy  $\wp = (\mathcal{F}, D, R, Q, \zeta)$  is

- *terminating* if for every  $q \in Q$ , all derivations of source  $q$  in  $\zeta$  are finite.  
*Every request evaluation terminates.*
- *consistent* if for every  $q \in Q$ ,  $\zeta$  applied to  $q$  returns at most one result:  $\forall q \in Q$ , the cardinality of  $\zeta q \cap D$  is less than or equal to 1.  
*For every request evaluation, at most one decision is computed.*
- *decision complete* if  $\forall q \in Q, \exists d \in D$ , such that  $d \in \zeta q$ .  
*Every request evaluation returns a decision.*

# Example - A firewall policy

<i>pkt</i>	:	<i>Address</i> × <i>Address</i> × <i>State</i>	↦	<i>Decision</i>
<i>new, est</i>	:		↦	<i>State</i>
<i>drop, accept</i>	:		↦	<i>Decision</i>
<i>eth0, ppp0, 10.1.1.1,</i> <i>10.1.1.2, 123.123.1.1</i>	:		↦	<i>Address</i>

1. *pkt(src, dst, est)* → *accept*
2. *pkt(eth0, dst, new)* → *accept*
3. *pkt(ppp0, dst, new)* → *drop*
4. *pkt(10.1.1.1, ppp0, s)* → *pkt(123.123.1.1, ppp0, s)*
5. *pkt(10.1.1.2, ppp0, s)* → *pkt(123.123.1.1, ppp0, s)*

Is it terminating ? consistent? decision complete?

# Example - A firewall policy

<i>pkt</i>	:	<i>Address</i> × <i>Address</i> × <i>State</i>	↦	<i>Decision</i>
<i>new, est</i>	:		↦	<i>State</i>
<i>drop, accept</i>	:		↦	<i>Decision</i>
<i>eth0, ppp0, 10.1.1.1,</i> <i>10.1.1.2, 123.123.1.1</i>	:		↦	<i>Address</i>

1. *pkt(src, dst, est)* → *accept*
2. *pkt(eth0, dst, new)* → *accept*
3. *pkt(ppp0, dst, new)* → *drop*
4. *pkt(10.1.1.1, ppp0, s)* → *pkt(123.123.1.1, ppp0, s)*
5. *pkt(10.1.1.2, ppp0, s)* → *pkt(123.123.1.1, ppp0, s)*

Is it terminating ? consistent? decision complete?

# Example - A firewall policy

<i>pkt</i>	:	<i>Address</i> × <i>Address</i> × <i>State</i>	↦	<i>Decision</i>
<i>new, est</i>	:		↦	<i>State</i>
<i>drop, accept</i>	:		↦	<i>Decision</i>
<i>eth0, ppp0, 10.1.1.1,</i> <i>10.1.1.2, 123.123.1.1</i>	:		↦	<i>Address</i>

1. *pkt(src, dst, est)* → *accept*
2. *pkt(eth0, dst, new)* → *accept*
3. *pkt(ppp0, dst, new)* → *drop*
4. *pkt(10.1.1.1, ppp0, s)* → *pkt(123.123.1.1, ppp0, s)*
5. *pkt(10.1.1.2, ppp0, s)* → *pkt(123.123.1.1, ppp0, s)*

Is it terminating ? consistent? decision complete?

## Example - A firewall policy

This policy is not decision complete:

*pkt(123.123.1.1, ppp0, new)???*

If the rule *pkt(123.123.1.1, ppp0, new) → accept* is added, the policy becomes decision complete.

1. *pkt(src, dst, est)* → *accept*
2. *pkt(eth0, dst, new)* → *accept*
3. *pkt(ppp0, dst, new)* → *drop*
4. *pkt(10.1.1.1, ppp0, s)* → *pkt(123.123.1.1, ppp0, s)*
5. *pkt(10.1.1.2, ppp0, s)* → *pkt(123.123.1.1, ppp0, s)*
6. *pkt(123.123.1.1, ppp0, new)* → *accept*

# Compositional challenge

- Rules describing various policies in large organizations are not authored and maintained in a single monolithic policy.
- The entities involved may have different, incompatible, or contradictory, requirements for access control.

In this context, relevant issues are:

- to provide mechanisms for combining policies with a uniform formal semantics
- to study how the interaction among the rules of multiple policies being combined can affect the general behavior of the global policy: are the properties of the component policies preserved?

- Rules describing various policies in large organizations are not authored and maintained in a single monolithic policy.
- The entities involved may have different, incompatible, or contradictory, requirements for access control.

In this context, relevant issues are:

- to provide mechanisms for combining policies with a uniform formal semantics
- to study how the interaction among the rules of multiple policies being combined can affect the general behavior of the global policy: are the properties of the component policies preserved?

See [Dougherty&all-Esorics07],[SantanaDeOliveiraPhD2008]

The *composition* of two policies  $\wp_i = (\mathcal{F}_i, D_i, R_i, Q_i, \zeta_i)$  ( $i = 1, 2$ ) is any policy  $\wp = (\mathcal{F}, D, R, Q, \zeta)$ , where:

- 1  $\mathcal{F}_1 \cup \mathcal{F}_2 \subseteq \mathcal{F}$ ;
- 2  $D_1 \cup D_2 \subseteq D \subseteq \mathcal{T}(\mathcal{F})$ ;
- 3  $R_1 \cup R_2 \subseteq R$ ;
- 4  $Q_1 \cup Q_2 \subseteq Q \subseteq \mathcal{T}(\mathcal{F})$ ;
- 5  $\zeta$  is a rewrite strategy for  $R$ .

- This definition provides simple union, sequential combination, conservative extension.
- Several existing approaches have been encoded using rewrite rules thanks to the expressivity provided by our strategy language. (*Ex: XACML-like Policy Combiners, majority*)
- Some modularity results are available with no specific strategy, or under innermost strategy.

# Example: XACML Combiners

- permit-overrides
- deny-overrides
- first-applicable

## Permit-Overrides

$po(\textit{permit}, y) \rightarrow \textit{permit}$   
 $po(x, \textit{permit}) \rightarrow \textit{permit}$   
 $po(\textit{deny}, na) \rightarrow \textit{deny}$   
 $po(na, \textit{deny}) \rightarrow \textit{deny}$   
 $po(na, na) \rightarrow na$

# Example: XACML Combiners

- permit-overrides
- deny-overrides
- first-applicable

## First-Applicable

$fa(\textit{permit}, y) \rightarrow \textit{permit}$

$fa(\textit{deny}, y) \rightarrow \textit{deny}$

$fa(\textit{na}, y) \rightarrow y$

# Example: Voting policy

The combination returns the decision of the majority of the sub-policies.

## Majority (permutative)

*maj(permit, permit, z) → permit*

*maj(deny, deny, z) → deny*

*maj(na, na, z) → na*

In which cases the combination is safe w.r.t consistency?

Simple case is when

- the policies do not share symbols
- the rule set is the union of the components' rewrite rules,
- there is no specific strategy.

The union of confluent rewrite systems is confluent [Toyama-87].

In which cases the combination is safe w.r.t consistency?

Simple case is when

- the policies do not share symbols
- the rule set is the union of the components' rewrite rules,
- there is no specific strategy.

The union of confluent rewrite systems is confluent [Toyama-87].

But with similar assumptions:

- the policies do not share symbols
  - the rule set is the union of the components' rewrite rules,
  - there is no specific strategy.
- 
- The union of terminating rewrite systems is not terminating [Toyama-87].
  - The union of two terminating rewrite systems is terminating under the *innermost* strategy [Gramlich-96].

But with similar assumptions:

- the policies do not share symbols
  - the rule set is the union of the components' rewrite rules,
  - there is no specific strategy.
- 
- The union of terminating rewrite systems is not terminating [Toyama-87].
  - The union of two terminating rewrite systems is terminating under the *innermost* strategy [Gramlich-96].

But with similar assumptions:

- the policies do not share symbols
  - the rule set is the union of the components' rewrite rules,
  - there is no specific strategy.
- 
- The union of terminating rewrite systems is not terminating [Toyama-87].
  - The union of two terminating rewrite systems is terminating under the *innermost* strategy [Gramlich-96].

Take advantage of the abstracting power of narrowing to schematize rewritings

- any narrowing derivation abstracts a family of rewriting derivations
- any rewriting derivation is captured by a narrowing derivation
- all requests to the policy are captured by this abstraction process.

# Simulation of rewriting by narrowing

Inside the machinery

For any term  $t \in \mathcal{T}(\mathcal{F}, \mathcal{Y})$  and rewrite system  $R$ ,  
if  $t \xrightarrow{\omega_1, l_1 \rightarrow r_1, \sigma_1}^R t_1 \cdots \xrightarrow{\omega_n, l_n \rightarrow r_n, \sigma_n}^R t_n$ , then

$$\sigma_n \cdots \sigma_1(t) \xrightarrow{\omega_1, l_1 \rightarrow r_1}^R \cdots \xrightarrow{\omega_n, l_n \rightarrow r_n}^R t_n$$

and for any ground instance  $\alpha$  of  $\sigma_n \cdots \sigma_1(t)$ ,

$$\alpha(t) \xrightarrow{\omega_1, l_1 \rightarrow r_1}^R \cdots \xrightarrow{\omega_n, l_n \rightarrow r_n}^R \alpha(t_n).$$

Let  $t_0$  be a term and  $\rho$  be an  $R$ -normalized substitution such that:

$$\rho(t_0) \xrightarrow{\omega_1, l_1 \rightarrow r_1} \cdots \xrightarrow{\omega_n, l_n \rightarrow r_n} t'_n.$$

Then there exist substitutions  $\sigma_i (i = 1, \dots, n)$  and  $\mu$  such that:

- 1  $t_0 \rightsquigarrow_{\omega_1, l_1 \rightarrow r_1, \sigma_1}^R t_1 \cdots \rightsquigarrow_{\omega_n, l_n \rightarrow r_n, \sigma_n}^R t_n$ ,
- 2  $\mu(t_n) = t'_n$ ,  $\mu$  is  $R$ -normalized,
- 3  $\rho = \text{var } t_0 \mu \sigma_n \cdots \sigma_1$  (i.e.  $\forall x \in \text{Var}(t_0), \rho(x) = \mu \sigma_n \cdots \sigma_1(x)$ ).

Whenever simulation of strategic rewriting by strategic narrowing can be established, we get tools for policy analysis:

- *Reachability analysis*: find all solutions of a reachability goal  $s \rightarrow^* t$ , i.e. substitutions  $\sigma$  such that  $\sigma(s) \rightarrow^* \sigma(t)$   
[MeseguerT-HOSC07]
- *What-if analysis*: solve queries of the form  $q(x_1, \dots, x_n)$
- *Exhaustive requests analysis*: find a correct and complete schematization of request evaluation.

Given a security policy  $\wp = (\mathcal{F}, D, R, Q, \zeta)$ , a *query* is a term of  $\mathcal{T}(\mathcal{F}, \mathcal{Y})$  where  $\mathcal{Y}$  is a set of query variables distinct from  $\mathcal{X}$ .

The method relies on the construction of a *narrowing tree* for a given query: all possible (strategic) narrowing steps are applied to this term and recursively to the resulting ones, yielding a possibly infinite tree.

*Termination results for narrowing?* [Hullot80,AlpuenteEI08]

# Example

*Which packets get accepted for a new connection?*

Solve the query  $\text{pkt}(x, y, \text{new})$ .

$\text{pkt}(x, y, \text{new})$				
$x = \text{eth0}$	$x = \text{ppp0}$	$x = 10.1.1.1$	$x = 10.1.1.2$	$x = 123.123.1.1$
$y = \text{dst}$	$y = \text{dst}$	$y = \text{ppp0}$	$y = \text{ppp0}$	$y = \text{ppp0}$
		$s = \text{new}$	$s = \text{new}$	
<b>accept</b>	<b>drop</b>	$\text{pkt}(123.123.1.1,$ $\text{ppp0}, \text{new})$	$\text{pkt}(123.123.1.1,$ $\text{ppp0}, \text{new})$	<b>accept</b>
		<b>accept</b>	<b>accept</b>	

So requests that give an *accept* decision are

$\langle \text{pkt}(\text{eth0}, \text{dst}, \text{new}) \rangle$  and  $\text{pkt}(10.1.1.1, \text{ppp0}, \text{new})$ ,  
 $\text{pkt}(10.1.1.2, \text{ppp0}, \text{new})$ ,  $\text{pkt}(123.123.1.1, \text{ppp0}, \text{new})$ .

# Exhaustive requests analysis

Assume that the policy  $\wp$  is terminating and decision complete.

A *decision term* is a ground term of  $D$  in normal form.

A *query pattern* is a term  $p(x_1, \dots, x_n)$ .

Assume that the set  $Q$  of requests is defined as the set of normalized ground instances of a finite set of *query patterns*.

The narrowing trees built from this finite set of query patterns provide a correct and complete schematization of the request evaluation.

# Example

*Which packets get accepted and which are dropped?*

Solve the query pattern  $\text{pkt}(x, y, z)$ .

**pkt(x, y, z)**

$x = \text{src}$     $x = \text{eth0}$     $x = \text{ppp0}$     $x = 10.1.1.1$   
 $y = \text{dst}$     $y = \text{dst}$     $y = \text{dst}$     $y = \text{ppp0}$   
 $z = \text{est}$     $z = \text{new}$     $z = \text{new}$     $z = s$

$x = 10.1.1.2$   
 $y = \text{ppp0}$   
 $z = s$

**accept**   **accept**   **drop**   **pkt(123.123.1.1, ppp0, s)**  
 $\text{src}' = 123.123.1.1$   
 $\text{dst}' = \text{ppp0}$   
 $s = \text{est}$

**pkt(123.123.1.1, ppp0, s)**  
 $\text{src}' = 123.123.1.1$   
 $\text{dst}' = \text{ppp0}$   
 $s = \text{est}$

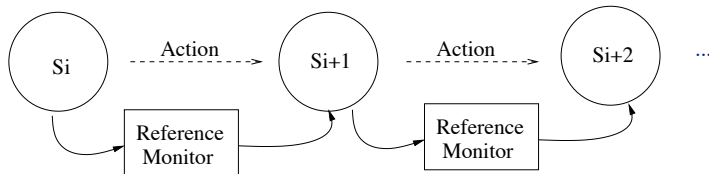
**accept**

**accept**

# Policy Enforcement Mechanism

How to enforce a policy in a given program?

- A reference monitor watches the execution of a target program, and interferes when it is about to violate the security policy
- Monitors can be “inlined” in the application code
- This process has been automated for low level policies



How to provide a methodology for building reference monitors for rewrite-based policies?

- Tom (<http://tom.loria.fr>)
- Extends Java with pattern matching, rewriting, and strategies
  - `match` allows discriminating among a list of patterns.
  - `'` (backquote construct) is used to build terms from Java values.
  - `strategy` groups rules for constructing more complex strategies, e.g. *innermost*, *outermost*, *top down*, etc..
  - `gom` provides a language to define tree-like data structures

[E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles.

**Tom: Piggybacking rewriting on java.** *RTA 2007*, vol. 4533 of *LNCS*, pages 36–47. Springer.]

- Tom (<http://tom.loria.fr>)
- Extends Java with pattern matching, rewriting, and strategies
  - `match` allows discriminating among a list of patterns.
  - `'` (backquote construct) is used to build terms from Java values.
  - `strategy` groups rules for constructing more complex strategies, e.g. *innermost*, *outermost*, *top down*, etc..
  - `gom` provides a language to define tree-like data structures

[E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles.

**Tom: Piggybacking rewriting on java.** *RTA 2007*, vol. 4533 of *LNCS*, pages 36–47.

Springer.]

- Tom (<http://tom.loria.fr>)
- Extends Java with pattern matching, rewriting, and strategies
  - `match` allows discriminating among a list of patterns.
  - `'` (backquote construct) is used to build terms from Java values.
  - `strategy` groups rules for constructing more complex strategies, e.g. *innermost*, *outermost*, *top down*, etc..
  - `gom` provides a language to define tree-like data structures

[E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles.

**Tom: Piggybacking rewriting on java.** *RTA 2007*, vol. 4533 of *LNCS*, pages 36–47.

Springer.]

- Tom (<http://tom.loria.fr>)
- Extends Java with pattern matching, rewriting, and strategies
  - `match` allows discriminating among a list of patterns.
  - `'` (backquote construct) is used to build terms from Java values.
  - `strategy` groups rules for constructing more complex strategies, e.g. *innermost*, *outermost*, *top down*, etc..
  - `gom` provides a language to define tree-like data structures

[E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles.

**Tom: Piggybacking rewriting on java.** *RTA 2007*, vol. 4533 of *LNCS*, pages 36–47.

Springer.]

- Tom (<http://tom.loria.fr>)
- Extends Java with pattern matching, rewriting, and strategies
  - `match` allows discriminating among a list of patterns.
  - `'` (backquote construct) is used to build terms from Java values.
  - `strategy` groups rules for constructing more complex strategies, e.g. *innermost*, *outermost*, *top down*, etc..
  - `gom` provides a language to define tree-like data structures

[E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles.

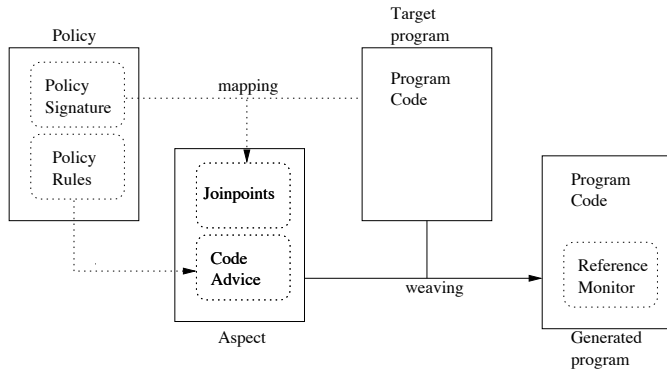
**Tom: Piggybacking rewriting on java.** *RTA 2007*, vol. 4533 of *LNCS*, pages 36–47.

Springer.]

# Aspect Oriented Programming

- Separation of “crosscutting” concerns, e.g. access control
- Encapsulation of behaviors that affect multiple classes in aspects
- Basic concepts: *joinpoints* and *code advice*
- *Weaving*: compilation process that matches the joinpoints against the actual code and inserts the code advice *before*, *after* or *around* a joinpoint.
- Several implementations exist, a popular one is AspectJ

# System Architecture



**Claim:** Verification of properties and query analysis improves the trust of a security administrator in a given policy, not only at the design stage, but also whenever a policy is updated.

## Related work:

Explore the information flow over the reachable states of a system and detect information leakage [Cirstea&all-Secret08].

## Future work concerns

- fine-tuned analysis to formalize and tackle the various strategies used by policy developers.
- modular policy analysis for large and distributed organizations. [Barker&all-Secret08].