

Modular Access Control via Strategic Rewriting

Daniel J. Dougherty¹, Claude Kirchner², Hélène Kirchner³, Anderson Santana de Oliveira²

Worcester Polytechnic Institute

INRIA & LORIA

CNRS & LORIA

SSURF, April, 2007

Motivation

- We want to give a formal semantics for flexible access control specification languages
- We want to prove properties of policies and policy composition using formal methods and tools
- Term rewriting provides us with a very suitable theoretical background to support both policy specification and composition, since we dispose of positive modular results on properties of rewrite system

Recent Contributions

- We extend previous work [de Oliveira, 2006] to support more general policies:
 - Allow policies with multiple possible decisions
 - Allow the definition of composition operator with clear semantics
- We base the semantics of policies and composition operators on strategic rewriting, which allows us to take advantage of the framework expressivity and to check properties of policies

Plan of the Talk

1 Introduction

2 Background

Introducing Strategies
Examples

3 Policy Composition

Composition Examples
Semantics of XACML Policy Combiners

4 Summary and Further Works

Defining Rewriting Strategies.

Definition

A *rewrite strategy* ζ for the rewrite system R is a subset of the set of all derivations of R .

The *application of a strategy* ζ on a term t is denoted $[\zeta](t)$ and defined as the set of all targets t' of the derivations of source t in ζ .

The *domain* of a strategy is the set of terms that are source of a derivation in ζ . When no derivation in ζ has for source t , we say that the strategy application on t fails

Defining Rewriting Strategies.

Definition

A *rewrite strategy* ζ for the rewrite system R is a subset of the set of all derivations of R .

The *application of a strategy* ζ on a term t is denoted $[\zeta](t)$ and defined as the set of all targets t' of the derivations of source t in ζ .

The *domain* of a strategy is the set of terms that are source of a derivation in ζ . When no derivation in ζ has for source t , we say that the strategy application on t fails

Defining Rewriting Strategies.

Definition

A *rewrite strategy* ζ for the rewrite system R is a subset of the set of all derivations of R .

The *application of a strategy* ζ on a term t is denoted $[\zeta](t)$ and defined as the set of all targets t' of the derivations of source t in ζ .

The *domain* of a strategy is the set of terms that are source of a derivation in ζ . When no derivation in ζ has for source t , we say that the strategy application on t fails

Strategy Application Operator.

The strategy application operator $@$, where the symbol $@$ represents placeholders for the arguments, respectively, an strategy, ζ , and a term, t , producing the object $[\zeta](t)$.

Example

If a and b are constants, the application of the rewrite rule $a \rightarrow b$ to the term a is denoted $[a \rightarrow b](a)$ and evaluates to $\{b\}$.

Strategy Languages.

A strategy could be described by enumerating all its elements or more suitably by a *strategy language*, e.g. ELAN [Kirchner et al., 1995], Stratego [Visser, 2001], Tom [Balland et al., 2006a] or MAUDE [Martí-Oliet et al., 2005] .

A simple strategy is the sequential application of two rules. It is described by the concatenation operator “seq”. For instance $[seq(l_1 \rightarrow r_1, l_2 \rightarrow r_2)](t)$ denotes $[l_2 \rightarrow r_2]([l_1 \rightarrow r_1](t))$. This strategy operator extends naturally to multiple arguments:

$$[seq(\zeta_1, \dots, \zeta_n)](t) = [\zeta_n]([\zeta_{n-1}]([\dots [\zeta_1](t)]))$$

Strategy Languages.

A strategy could be described by enumerating all its elements or more suitably by a *strategy language*, e.g. ELAN [Kirchner et al., 1995], Stratego [Visser, 2001], Tom [Balland et al., 2006a] or MAUDE [Martí-Oliet et al., 2005] .

A simple strategy is the sequential application of two rules. It is described by the concatenation operator “seq”. For instance $[seq(l_1 \rightarrow r_1, l_2 \rightarrow r_2)](t)$ denotes $[l_2 \rightarrow r_2]([l_1 \rightarrow r_1](t))$. This strategy operator extends naturally to multiple arguments:

$$[seq(\zeta_1, \dots, \zeta_n)](t) = [\zeta_n]([\zeta_{n-1}]([\dots [\zeta_1](t)]))$$

Strategy Operators.

Basic Strategies.

$$\begin{aligned}[\text{id}](t) &= \{t\} \\ [\text{fail}](t) &= \emptyset\end{aligned}$$

universal.

The strategy computing all derivations issued from the application of a rewrite system R .

$$[\text{universal}(R)](t) = \{t' \mid t \xrightarrow{*}_R t'\}$$

Example

For instance, we have:

$$\begin{aligned}[\text{universal}(a \rightarrow a)](a) &= \{a\} \\ [\text{universal}(f(x) \rightarrow f(f(x)))](f(a)) &= \{f(a), f(f(a)), f(f(f(a))), \dots\}\end{aligned}$$

Choice Operator.

The Choice Strategy.

$$\begin{aligned} [\text{choice}(\zeta_1, \zeta_2)](t) &= [\zeta_1](t) && \text{if } [\zeta_1](t) \neq \emptyset \\ [\text{choice}(\zeta_1, \zeta_2)](t) &= [\zeta_2](t) && \text{if } [\zeta_1](t) = \emptyset \end{aligned}$$

Clearly `choice` is associative and therefore its syntax is extended to be applicable to a list of strategies:

$$\text{choice}(\zeta_1, \zeta_2, \dots, \zeta_n) = \text{choice}(\zeta_1, \text{choice}(\zeta_2, \dots, \zeta_n))$$

Other Useful Strategies.

One/All.

$$[\text{one}(\zeta)](f(t_1, \dots, t_n)) = f(t_1, \dots, [\zeta](t_i), \dots, t_n), \text{ if } [\zeta](t_i) \neq \emptyset$$

$$[\text{all}(\zeta)](f(t_1, \dots, t_n)) = f([\zeta](t_1), \dots, [\zeta](t_n)), \text{ if } \forall i \in \{1, \dots, n\}, [\zeta](t_i) \neq \emptyset$$

Try/Repeat.

$$\text{try}(\zeta) = \text{choice}(\zeta, \text{id})$$

$$\text{repeat}(\zeta) = \text{try}(\text{seq}(\zeta, \text{repeat}(\zeta)))$$

Traversal Strategies.

Some Standard Strategies in Rewriting.

$\text{topDown}(\zeta)$	$= \text{seq}(\zeta, \text{all}(\text{topDown}(\zeta)))$
$\text{bottomUp}(\zeta)$	$= \text{seq}(\text{all}(\text{bottomUp}(\zeta)), \zeta)$
$\text{OnceTopDown}(\zeta)$	$= \text{choice}(\zeta, \text{one}(\text{OnceTopDown}(\zeta)))$
$\text{OnceBottomUp}(\zeta)$	$= \text{choice}(\text{one}(\text{OnceBottomUp}(\zeta)), \zeta)$
$\text{innermost}(\zeta)$	$= \text{repeat}(\text{onceBottomUp}(\zeta))$
$\text{outermost}(\zeta)$	$= \text{repeat}(\text{onceTopDown}(\zeta))$

Some Strategy Application Examples

Example

Some examples of strategy application are:

$$\begin{aligned}[\text{universal}(a \rightarrow b, a \rightarrow c)](a) &= \{a, b, c\} \\ [\text{choice}(a \rightarrow b, a \rightarrow c)](a) &= \{b\} \\ [\text{choice}(a \rightarrow c, a \rightarrow b)](b) &= \emptyset \\ [\text{try}(b \rightarrow c)](a) &= \{a\} \\ [\text{repeat}(\text{choice}(b \rightarrow c, a \rightarrow b))](a) &= \{c\}\end{aligned}$$

Rewriting-Based Policies.

We define rewriting-based policies as follows.

Definition (Security Policy)

An access control security policy, \wp , is a 5-tuple $(\mathcal{F}, D, R, Q, \zeta)$ such that:

- 1 \mathcal{F} is a signature;
- 2 D is a non-empty set of ground terms: $D \subseteq \mathcal{T}(\mathcal{F})$;
- 3 R is a set of rewrite rules over $\mathcal{T}(\mathcal{F}, \mathcal{X})$;
- 4 Q is a set of terms from $\mathcal{T}(\mathcal{F})$: $Q \subseteq \mathcal{T}(\mathcal{F})$;
- 5 ζ is a rewrite strategy for R .

Discussion.

- The policy specification and its environment are described as terms built over the signature \mathcal{F}
- D is often a set of constants and the two main constants in D are usually *permit* and *deny*. The result returned by a policy could be more elaborated than just a constant and can be a ground term containing further information.
- What is significant for composition is not treating the failure to derive a permission as a denial.
- The rewrite system R describes the behavior of the policy as well as some necessary computations which explain how its environment evolves. The role of the strategy is to point derivations of R whose interest is to produce decisions.
- The requests are a subset of ground terms. They express whether an entity authorized to access a resource given the current configuration of the policy environment.
- The last component is the strategy which allows one to finely specify the evaluation order of the policy rules.

Basic Example.

Assigns authorizations based on a “user id” which is represented by a natural number: all requests from user whose “id” is bigger than three are denied.

- Let the policy signature be:
 $\mathcal{F} = \{0 : Nat, s : Nat \rightarrow Nat, + : Nat \times Nat \rightarrow Nat, auth : Nat \rightarrow A, permit : A, na : A, deny : A\}$
- The set of of constant symbols representing decisions is $D = \{permit, na, deny\}$
- Consider R as the following set of rules (the operator s gives the successor of a numnber, $+$ is the usual sum operator, x, y are variables of sort Nat):

$$\begin{array}{ll}
 x + s(y) & \rightarrow s(x + y) \\
 x + 0 & \rightarrow x \\
 auth(0) & \rightarrow permit \\
 auth(s(0)) & \rightarrow permit \\
 auth(s(s(0))) & \rightarrow na \\
 auth(s(s(s(x)))) & \rightarrow deny
 \end{array}$$

Basic Example.

- the set Q contains ground terms with top symbol *auth*;
- A possible strategy for this policy, among others that guarantee a normalization process, is $\zeta = \text{innermost}(R)$.

This defines a security police as all conditions of Definition 5 are satisfied. An example of request evaluation is:

$$[\zeta](\text{auth}(s(0) + s(s(s(0)))))) = \{\text{deny}\}$$

Basic Example.

- the set Q contains ground terms with top symbol *auth*;
- A possible strategy for this policy, among others that guarantee a normalization process, is $\zeta = \text{innermost}(R)$.

This defines a security police as all conditions of Definition 5 are satisfied. An example of request evaluation is:

$$[\zeta](\text{auth}(s(0) + s(s(s(0)))))) = \{\text{deny}\}$$

Basic Policy Properties.

Definition (Consistency)

A security policy $\wp = (\mathcal{F}, D, R, Q, \zeta)$ is consistent if for every query $q \in Q$, ζ applied to q returns at most one result:

$\forall q \in Q$, the cardinality of $[\zeta](q)$ is less than or equal to 1.

For every query evaluation, a deterministic result is computed by the application of ζ on the terms of Q .

In the case where the strategy leads to a derivation that does not terminate on q , the cardinality of $[\zeta](q)$ is 0, the policy is still considered as consistent.

Basic Policy Properties.

Definition (Consistency)

A security policy $\wp = (\mathcal{F}, D, R, Q, \zeta)$ is consistent if for every query $q \in Q$, ζ applied to q returns at most one result:

$\forall q \in Q$, the cardinality of $[\zeta](q)$ is less than or equal to 1.

For every query evaluation, a deterministic result is computed by the application of ζ on the terms of Q .

In the case where the strategy leads to a derivation that does not terminate on q , the cardinality of $[\zeta](q)$ is 0, the policy is still considered as consistent.

Inconsistent Policy.

Example

Consider the following policy:

$$\wp_1 = (\begin{array}{l} \mathcal{F}_1 = \{g : A \times A \rightarrow A, \textit{permit} : A, \textit{deny} : A\}, \\ D_1 = \{\textit{permit}, \textit{deny}\}, \\ R_1 = \{g(x, y) \rightarrow x, g(x, y) \rightarrow y\}, \\ Q_1 = g(\mathcal{T}(\mathcal{F}), \mathcal{T}(\mathcal{F})), \\ \zeta_1 = \textit{universal}(R) \end{array})$$

Then \wp_1 is a security policy under the conditions expressed in Definition 5, but it clearly fails to be consistent, since $[\zeta](g(\textit{permit}, \textit{deny})) = \{\textit{permit}, \textit{deny}\}$.

Consistency.

Definition (Confluence under strategy)

A rewrite system R is confluent under a strategy ζ when $\forall u, v_1, v_2 \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $\{v_1, v_2\} \subseteq [\zeta](u)$ then $[\zeta](v_1) \cap [\zeta](v_2) \neq \emptyset$.

If we consider the `universal` strategy, the above definition reduces to the usual one of confluence. Therefore:

Definition

The policy $(\mathcal{F}, D, R, Q, \text{universal})$ is consistent as soon as R is confluent on $\mathcal{T}(\mathcal{F}, \mathcal{X})$.

Termination.

Definition (Termination)

A security policy $\wp = (\mathcal{F}, D, R, Q, \zeta)$ is terminating if for every $q \in Q$, all derivations of source q in ζ are finite.

Non-terminating Policy.

Example

Consider the policy:

$$\begin{aligned} \wp_2 = (\quad & \mathcal{F}_2 = \{a : A, \textit{permit} : A, \textit{deny} : A\}, \\ & D_2 = \{\textit{permit}, \textit{deny}\}, \\ & R_2 = \{a \rightarrow a, a \rightarrow \textit{deny}\}, \\ & Q_2 = \{a\}, \\ & \zeta_2 = \text{universal}(R) \end{aligned}$$

\wp_2 is a security policy. In contrast to the previous example, this policy is consistent (since the corresponding rewrite relation is confluent), but it is not terminating.

Termination.

Definition

A policy $(\mathcal{F}, D, R, Q, \zeta)$ terminates provided that all derivations in ζ are finite or if R is strongly terminating (i.e. all derivations in $\text{universal}(R)$ are finite).

To ensure strong termination, classical quite powerful termination tools can be used like recursive path orderings [Dershowitz, 1987] or dependency pairs [Arts and Giesl, 2000].

Completeness.

Definition (Completeness)

A security policy $\wp = (\mathcal{F}, D, R, Q, \zeta)$ is complete if $\forall q \in Q$, $[\zeta](q) \subseteq D$ and $[\zeta](q) \neq \emptyset$.

This definition is close to the definition of sufficient completeness of a rewrite system, which states that every ground term evaluates to a term exclusively built with constructors and possibly variables [Comon, 1986, Kapur et al., 1991].

A More Elaborate Example - II

Based on the model proposed in [de Oliveira, 2006].

$$\mathcal{F} = \left\{ \begin{array}{l} \mathit{auth} : Q \times \mathit{Phase} \times \mathit{Assignment} \rightarrow A, \\ \mathit{q} : \mathit{Subject} \times \mathit{Action} \times \mathit{Object} \rightarrow Q, \\ \mathit{submission}, \mathit{meeting}, \mathit{review} : \mathit{Phase}, \\ \mathit{submitPaper}, \mathit{readScores}, \mathit{submitReview} : \mathit{Action}, \\ \mathit{author}, \mathit{reviewer}, \mathit{chair} : \mathit{Name} \rightarrow \mathit{Subject}, \\ \mathit{paper} : \mathit{Name} \times \mathit>Title \rightarrow \mathit{Object}, \\ \mathit{assigned} : \mathit{Name} \times \mathit{Object} \rightarrow \mathit{Assignment}, \end{array} \right. \cup$$
$$D = \{ \mathit{permit} : A, \mathit{deny} : A, \mathit{na} : A \}$$

A More Elaborate Example - III

The requests are represented by the terms rooted by the function symbol:

$$Q = \mathit{auth} : Q \times \mathit{Phase} \times \mathit{Assignment} \rightarrow A$$

R is the following set of rules

$$\mathit{auth}(q(\mathit{author}(x), \mathit{submitPaper}, \mathit{paper}(x, y)), \mathit{submission}, z) \rightarrow \mathit{permit}$$

$$\mathit{auth}(q(\mathit{reviewer}(x), \mathit{submitReview}, \mathit{paper}(y, z)), \mathit{review}, \mathit{assigned}(x, \mathit{paper}(y, z))) \rightarrow \mathit{permit}$$

$$\mathit{auth}(q(\mathit{reviewer}(x), \mathit{readScores}, \mathit{paper}(y, z)), \mathit{meeting}, \mathit{assigned}(x, \mathit{paper}(y, z))) \rightarrow \mathit{permit}$$

$$\mathit{auth}(q(\mathit{author}(x), \mathit{readScores}, \mathit{paper}(x, y)), w, z) \rightarrow \mathit{deny}$$

$$\zeta = \mathit{choice}(R, \mathit{auth}(x, y, z) \rightarrow na)$$

A More Elaborate Example - III

The requests are represented by the terms rooted by the function symbol:

$$Q = \text{auth} : Q \times \text{Phase} \times \text{Assignment} \rightarrow A$$

R is the following set of rules

$$\text{auth}(q(\text{author}(x), \text{submitPaper}, \text{paper}(x, y)), \text{submission}, z) \rightarrow \text{permit}$$

$$\text{auth}(q(\text{reviewer}(x), \text{submitReview}, \text{paper}(y, z)), \text{review}, \text{assigned}(x, \text{paper}(y, z))) \rightarrow \text{permit}$$

$$\text{auth}(q(\text{reviewer}(x), \text{readScores}, \text{paper}(y, z)), \text{meeting}, \text{assigned}(x, \text{paper}(y, z))) \rightarrow \text{permit}$$

$$\text{auth}(q(\text{author}(x), \text{readScores}, \text{paper}(x, y)), w, z) \rightarrow \text{deny}$$

$$\zeta = \text{choice}(R, \text{auth}(x, y, z) \rightarrow na)$$

Policy Composition Strategies.

Definition (Policy Composition)

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

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

Discussion.

- When defining the composition of two policies, we must ensure that the generated policy satisfies the conditions declared in Definition 5
- The set of requests for the combined policy contains terms of the form determined by its sub-policies, but may also contain any additional well-formed closed term that can be constructed from the combined policy signature. For example, suppose that $\mathcal{F}_1 = \{0, f\}$, $Q_1 = f(\mathcal{I}(\mathcal{F}_1))$ and $\mathcal{F}_2 = \{g\}$, $Q_2 = g(\mathcal{I}(\mathcal{F}_2))$, then a valid request would be $g(f(0))$;
- The combination strategy is in charge of defining how the composed policy rewrites request terms. It may or not be built in a modular way by composing ζ_1 and ζ_2 .

Extending the Conference Policy.

Example

The policy signature has been extended with the function symbol $conflict : Name \times Object \rightarrow Assignment$, and the following set of rules, R' :

$$\begin{aligned} &auth(\quad q(\text{reviewer}(r), \text{submitReview}, p), \text{phase}, \text{conflict}(r, p)) \\ &\quad \rightarrow \text{deny}, \\ &auth(\quad q(\text{reviewer}(n), \text{readScores}, p), \text{phase}, \text{conflict}(n, p)) \\ &\quad \rightarrow \text{deny} \end{aligned}$$

A strategy

$\zeta' = \text{choice}(\text{choice}(R', R), \text{auth}(q(x, y, z), w, k) \rightarrow na)$
assures it is still a security policy.

A Second Example.

Example

Consider the policies \wp_1 and \wp_2 below.

$$\wp_1 = (\begin{array}{l} \mathcal{F}_1 = \{g : A \times A \rightarrow A\} \cup D = \{\textit{permit}, \textit{deny}, \textit{na} : A\}, \\ P_1 = \{g(x, y) \rightarrow x, g(x, y) \rightarrow y\}, \\ Q_1 = g : A \times A \rightarrow A, \\ \zeta_1 = \textit{universal}(P_1) \end{array})$$

$$\wp_2 = (\begin{array}{l} \mathcal{F}_2 = \{f : A \times A \times A \rightarrow A\} \cup D = \{\textit{permit}, \textit{deny}, \textit{na} : A\} \\ P_2 = \{f(\textit{permit}, \textit{deny}, x) \rightarrow f(x, x, x), \\ f(\textit{deny}, \textit{permit}, x) \rightarrow f(x, x, x), \\ f(x, x, x) \rightarrow x\}, \\ Q_2 = f : A \times A \times A \rightarrow A, \\ \zeta_2 = \textit{universal}(P_2) \end{array})$$

A Second Example.

The composition \wp of \wp_1 and \wp_2 can be defined in a straightforward way as $\wp =$:

$$(\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_3, D = D_1 = D_3, R = R_1 \cup R_3, Q = T(\mathcal{F}_1 \cup \mathcal{F}_3), \zeta = \text{universal}(R))$$

These two policies are clearly terminating and share only symbols *permit* and *deny*. It is therefore quite intuitive to believe that their composition will be also terminating. But this is false since the following request has an infinite derivation:

$$\begin{aligned} & f(g(\text{permit}, \text{deny}), g(\text{permit}.\text{deny}), g(\text{permit}, \text{deny})) \rightarrow \\ & f(\text{permit}, g(\text{permit}, \text{deny}), g(\text{permit}, \text{deny})) \rightarrow \\ & f(\text{permit}, \text{deny}, g(\text{permit}, \text{deny})) \rightarrow \\ & f(g(\text{permit}, \text{deny}), g(\text{permit}.\text{deny}), g(\text{permit}, \text{deny})) \dots \end{aligned}$$

A Second Example.

The composition \wp of \wp_1 and \wp_2 can be defined in a straightforward way as $\wp =$:

$$(\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_3, D = D_1 = D_3, R = R_1 \cup R_3, Q = T(\mathcal{F}_1 \cup \mathcal{F}_3), \zeta = \text{universal}(R))$$

These two policies are clearly terminating and share only symbols *permit* and *deny*. It is therefore quite intuitive to believe that their composition will be also terminating. But this is false since the following request has an infinite derivation:

$$\begin{aligned} & f(g(\text{permit}, \text{deny}), g(\text{permit}.\text{deny}), g(\text{permit}, \text{deny})) \rightarrow \\ & f(\text{permit}, g(\text{permit}, \text{deny}), g(\text{permit}, \text{deny})) \rightarrow \\ & f(\text{permit}, \text{deny}, g(\text{permit}, \text{deny})) \rightarrow \\ & f(g(\text{permit}, \text{deny}), g(\text{permit}.\text{deny}), g(\text{permit}, \text{deny})) \dots \end{aligned}$$

A Second Example.

The composition \wp of \wp_1 and \wp_2 can be defined in a straightforward way as $\wp =$:

$$(\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_3, D = D_1 = D_3, R = R_1 \cup R_3, Q = T(\mathcal{F}_1 \cup \mathcal{F}_3), \zeta = \text{universal}(R))$$

These two policies are clearly terminating and share only symbols *permit* and *deny*. It is therefore quite intuitive to believe that their composition will be also terminating. But this is false since the following request has an infinite derivation:

$$\begin{aligned} & f(g(\text{permit}, \text{deny}), g(\text{permit}.\text{deny}), g(\text{permit}, \text{deny})) \rightarrow \\ & f(\text{permit}, g(\text{permit}, \text{deny}), g(\text{permit}, \text{deny})) \rightarrow \\ & f(\text{permit}, \text{deny}, g(\text{permit}, \text{deny})) \rightarrow \\ & f(g(\text{permit}, \text{deny}), g(\text{permit}.\text{deny}), g(\text{permit}, \text{deny})) \dots \end{aligned}$$

Modularity Results

Many modularity results for confluence and termination of rewrite systems have been produced and the interested reader can refer for instance to [Ohlebusch, 2002] for a survey.

Definition

Let us consider two policies $\wp_i = (\mathcal{F}_i, D_i, R_i, Q_i, \text{universal})$ ($i = 1, 2$) such that \mathcal{F}_1 and \mathcal{F}_2 are disjoint and their composition $\wp = (\mathcal{F}_1 \cup \mathcal{F}_2, D_1 \cup D_2, R_1 \cup R_2, \mathcal{T}(\mathcal{F}_1 \cup \mathcal{F}_2), \text{universal})$. If \wp_1 and \wp_2 are consistent, then \wp is consistent. If \wp_1 and \wp_2 are terminating, then so is \wp , provided:

- 1 *neither R_1 nor R_2 contain collapsing rules, or*
- 2 *neither R_1 nor R_2 contain duplicating rules, or*
- 3 *R_1 or R_2 contains neither collapsing rules nor duplicating rules, or*
- 4 *termination of R_1 and of R_2 are proved by simplification ordering.*

Modularity Results

Definition

Let us consider two policies

$\wp_i = (\mathcal{F}_i, D_i, R_i, Q_i, \text{innermost})$ ($i = 1, 2$) *such that \mathcal{F}_1 and \mathcal{F}_2 are disjoint or share only constructors, and \wp be their composition*

$(\mathcal{F}_1 \cup \mathcal{F}_2, D_1 \cup D_2, R_1 \cup R_2, \mathcal{T}(\mathcal{F}_1 \cup \mathcal{F}_2), \text{innermost})$. *Then \wp is terminating as soon as \wp_1 and \wp_2 are.*

A Second Example

Example

Consider the policies \wp_1 and \wp_2 below.

$$\wp_1 = (\begin{array}{l} \mathcal{F}_1 = \{g : A \times A \rightarrow A\} \cup D = \{\text{permit}, \text{deny}, \text{na} : A\}, \\ P_1 = \{g(x, y) \rightarrow x, g(x, y) \rightarrow y\}, \\ Q_1 = g : A \times A \rightarrow A, \\ \zeta_1 = \text{innermost}(P_1) \end{array})$$

$$\wp_2 = (\begin{array}{l} \mathcal{F}_2 = \{f : A \times A \times A \rightarrow A\} \cup D = \{\text{permit}, \text{deny}, \text{na} : A\} \\ P_2 = \{f(\text{permit}, \text{deny}, x) \rightarrow f(x, x, x), \\ f(\text{deny}, \text{permit}, x) \rightarrow f(x, x, x), \\ f(x, x, x) \rightarrow x\}, \\ Q_2 = f : A \times A \times A \rightarrow A, \\ \zeta_2 = \text{innermost}(P_2) \end{array})$$

Their combination

$\wp = (\mathcal{F}_1 \cup \mathcal{F}_2, D, R_1 \cup R_2, \mathcal{T}(\mathcal{F}_1 \cup \mathcal{F}_2), \text{innermost}(R))$ is terminating according to Proposition 4.

XACML Combinators

- *permit-overrides*: whenever *one* of the policies answers to a request with a *granting* decision, the final authorization for the composed policy will be granted. The policy will generate a *denial* only in the case at least one of the sub-policies denies the request, and all others return *not-applicable*.
- *deny-overrides*: this combiner has a similar semantics to *permit-overrides*, with the difference that denials takes precedence.
- *first-applicable*: the decision produced by the combined policy corresponds to the authorization determined by the first sub-policy that does not fail, and whose decision is different from *not-applicable*.

The semantics of XACML Combinators

Permit Overrides:

$$[\zeta_{po}(\zeta_1, \zeta_2)](q) = \begin{cases} \{permit\} & \text{if } ([\zeta_1](q) = \{permit\} \vee [\zeta_2](q) = \{permit\}) \\ \{deny\} & \text{if } ([\zeta_1](q) = \{deny\} \vee [\zeta_2](q) = \{deny\}) \\ & \wedge ([\zeta_1](q) = \{na\} \vee [\zeta_2](q) = \{na\}) \\ \{na\} & \text{if } ([\zeta_1](q) = \{na\} \wedge [\zeta_2](q) = \{na\}) \end{cases}$$

The semantics of XACML Combinators

Permit Overrides Alternatively:

$$[\zeta_{po}(\zeta_1, \zeta_2)](q) = \text{choice}(\text{seq}(\zeta_1(q), \text{permit} \rightarrow \text{permit}), \text{seq}(\zeta_2(q), \text{permit} \rightarrow \text{permit}), \text{seq}(\zeta_1(q), \text{deny} \rightarrow \text{deny}), \text{seq}(\zeta_2(q), \text{deny} \rightarrow \text{deny}), \zeta_1(q), \zeta_2(q))$$

The semantics of XACML Combinators

The strategy *first-applicable* can be encoded in a similar fashion:

$$[\zeta_{fa}(\zeta_1, \zeta_2)](q) = \text{choice}(\text{seq}(\zeta_1(q), \text{permit} \rightarrow \text{permit}), \text{seq}(\zeta_1(q), \text{deny} \rightarrow \text{deny}), \text{seq}(\zeta_2(q), \text{permit} \rightarrow \text{permit}), \text{seq}(\zeta_2(q), \text{deny} \rightarrow \text{deny}), \zeta_1(q), \zeta_2(q))$$

Perspectives, Ongoing Work

Our main contributions:

- A formal definition for access control policies using term rewriting that allows us to describe flexible policies;
- To support reasoning about properties like consistency, completeness and termination;
- To give the formal semantics for composition operators in an uniform manner using rewrite strategies.

For future work we have the following goals:

- Provide an implementation of a rewriting-based policy language as a Formal Islands [Balland et al., 2006b].

For Further Information



Arts, T. and Giesl, J. (2000).

Termination of term rewriting using dependency pairs.
Theoretical Computer Science, 236:133–178.



Balland, E., Brauner, P., Kopetz, R., Moreau, P.-E., and Reilles, A. (2006a).
Tom Manual.

LORIA, Nancy (France), version 2.4 edition.



Balland, E., Kirchner, C., and Moreau, P.-E. (2006b).

Formal islands.

In Johnson, M. and Vene, V., editors, *AMAST*, volume 4019 of *Lecture Notes in Computer Science*, pages 51–65. Springer.



Comon, H. (1986).

Sufficient completeness, term rewriting systems and "anti-unification".

In Siekmann, J. H., editor, *CADE*, volume 230 of *Lecture Notes in Computer Science*, pages 128–140. Springer.



de Oliveira, A. S. (2006).

Rewriting-based access control policies.

In Fernandez, M. and Kirchner, C., editors, *Proceedings of the 1st International Workshop on Security and Rewriting Techniques - SecRet'06*.



Dershowitz, N. (1987).

Termination of rewriting.

Journal of Symbolic Computation, 3(1 & 2):69–116.



Kapur, D., Narendran, P., Rosenkrantz, D. J., and Zhang, H. (1991).

Sufficient-completeness, ground-reducibility and their complexity.
Acta Inf., 28(4):311–350.



Kirchner, C., Kirchner, H., and Vittek, M. (1995).

Designing clp using computational systems.