

Replace this file with `prentcsmacro.sty` for your meeting,  
or with `entcsmacro.sty` for your meeting. Both can be  
found at the [ENTCS Macro Home Page](#).

# Analysis of Rewrite-Based Access Control Policies

Claude Kirchner, H el ene Kirchner

*INRIA Bordeaux - Sud-Ouest Research Center  
351, Cours de la lib eration, 33405 Talence, France*

Anderson Santana de Oliveira<sup>1</sup>

*Universidade Federal do Rio Grande do Norte  
DIMAp - Campus Universit ario - Lagoa Nova  
59072-970, Natal-RN, Brazil*

---

Abstract

The rewrite-based approach provides executable specifications for security policies, which can be independently designed, verified, and then anchored on programs using a modular discipline. In this paper, we describe how to perform queries over these rule-based policies in order to increase the trust of the policy author on the correct behavior of the policy. The analysis we provide is founded on the narrowing process, which provides both the necessary abstraction for simulating executions of the policy over access requests and the mechanism for solving *what-if* queries from the security administrator. We illustrate this general approach by the analysis of a firewall system policy.

*Keywords:* Security Policies, Term Rewriting Systems, strategic rewriting, strategic narrowing.

---

## 1 Introduction

Security policies define what it means to be secure for a system. Policies establish constraints on how data are to be accessed, either by determining acceptable information flows, or by describing the privileges principals have over the protected resources inside the system. Since policies reflect the security requirements for some organization, or generally speaking, for some entity, they are subject to frequent changes as new threats appear or as the architecture of the system in question evolves. One of the current challenges in computer security is to model rich, expressive policies, usually given as a set of rules, such that their properties can be formally stated and proved. For instance, policies should be non-ambiguous, which means that an access is not granted and denied at the same time. Policies should

---

<sup>1</sup> Supported by CNPq 152373/2007-1.

also cover all relevant situations a system may be exposed to: when dealing with access control, this means answering all possible access requests.

In [13,9] we proposed a formal model of policies, based on term rewriting, which provides several advantages: first, the language allows us to handle a wide range of security policies, because we can easily describe the form of the access requests and the set of possible authorization decisions, without restricting them to simply *permit* or *deny*. Moreover, policy application can be defined in a precise and expressive way, since it is possible to determine a strategy to control rule application.

Such an approach provides not only a clear semantics to access control policies, but also appropriate techniques to verify important properties, relying on the confluence, termination and sufficient completeness of the underlying rewrite systems. Moreover, the rewrite-based framework inherits from the modularity results for all these properties, which is a striking advantage over other frameworks, since one can foresee the impact of the policy composition over the properties of the component policies.

Besides proving these properties, a policy designer needs to understand how access decisions are generated. He may want to know how a policy deals with specific kinds of requests, specially those concerning the most sensitive information in the system. In the literature, this is often referred to as “administrator queries” [25,6], representing questions of the kind “what if a request is made under these conditions? Will access be conceded?” Answering these questions increases the confidence of the policy designer in a given policy specification.

In this paper, we build on our previous works to provide this kind of analysis for rewrite-based policies. The main mechanism behind the analysis is narrowing, which provides both the necessary abstraction for simulation of rewriting-based executions of the policy requests and the solving mechanism of queries. We extend the policy language with requests involving variables and we show how narrowing can be used to find which values can instantiate such variables in order to satisfy given requests, thus providing an adequate mechanism for solving these queries.

Due to the expressiveness and computational power of rewriting, the framework of rewrite-based security policies is general enough to be applied to a large variety of security problems. An illustrative example is provided by firewall policies. A firewall is a security component put in the entry point of a local network to control its interactions with the Internet. The main objective is to inspect the packet traffic from and into the local network and to decide whether a given packet should be transmitted or rejected. Real firewall implementations are usually given as a sequence (i.e. totally ordered set) of rules, which can make recursive calls, and may have default rules. This is the case for instance for NetFilter, which is the firewall system used in several variations of Linux distributions. It is important to notice that such a rule presentation is used as a description of the rule execution sequence. This is indeed an implicit rule execution strategy employed by these systems, and the capacity of expressing such strategies explicitly is primordial to make clear, stable, communicable, maintainable, trustable and provable the policy semantics. In current systems, security administrators often face the problem of avoiding an inconsistent, redundant or incomplete set of rules as described for example in [22].

Here, we show how solving queries can be useful to identify such situations for

this extremely important kind of policy, since they are the most current mechanism for protecting networks from numerous threats. But the technique is also applicable for any other form of flexible rewrite-based policy.

The structure of the paper is as follows. In the next Section 2, we recall and illustrate the definition of a rewrite-based security policy. In Section 3, the relevant definitions of rewriting, narrowing and strategies are given. It stresses the notion of strategic rewriting which is of prime interest in the context of policy rules to take into account their order of application and more generally their control. Then Section 4 shows how to use strategic narrowing to solve queries and find the corresponding requests in a large class of policy specifications. It provides examples of what-if analysis and decisions analysis. To support the reader intuition, a simplified firewall example is developed along the paper. Section 5 concludes with some perspectives for further work.

## 2 Rewrite-Based Policies

The following standard notations will be used in the following.  $\mathcal{T}(\mathcal{F}, \mathcal{X})$  is the set of first-order terms built from a given finite set  $\mathcal{F}$  of function symbols with their profile declarations and a denumerable set  $\mathcal{X}$  of variables. Positions in a term are represented as sequences of integers and denoted by  $\omega$ , while the empty sequence  $\epsilon$  denotes the top position. The set of variables occurring in a term  $t$  is denoted by  $\mathcal{V}ar(t)$ . If  $\mathcal{V}ar(t)$  is empty,  $t$  is called a *ground term* and  $\mathcal{T}(\mathcal{F})$  is the set of ground terms. A substitution  $\sigma$  is an assignment from  $\mathcal{X}$  to  $\mathcal{T}(\mathcal{F}, \mathcal{X})$ , with a finite domain  $\{x_1, \dots, x_k\}$  and written  $\sigma = \{x_1 \mapsto t_1, \dots, x_k \mapsto t_k\}$ . Basic definitions on term rewriting can be found in [27,29,3]. A rewrite rule is an ordered pair of terms  $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ , denoted  $l \rightarrow r$ , where it is often required that  $l$  is not a variable and  $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ . The terms  $l$  and  $r$  are respectively called the left-hand side and the right-hand side of the rule. A rewrite system is a (finite or infinite) set of rewrite rules. Rules can be labeled to easily talk about them.

In [13], we introduced a general definition of an access control policy that abstracts the set of possible requests and decisions, and thus supports a wide range of policies:

**Definition 2.1** [Security Policy [13]] An access control security policy  $\wp$  is a 5-tuple  $(\mathcal{F}, D, R, Q, \zeta)$  where:

- (i)  $\mathcal{F}$  is a sorted signature; *to define the vocabulary of the data-structure.*
- (ii)  $D$  is a non-empty set of closed terms:  $D \subseteq \mathcal{T}(\mathcal{F})$ ; *to formalize the set of decisions taken by the policy.*
- (iii)  $R$  is a set of rewrite rules over  $\mathcal{T}(\mathcal{F}, \mathcal{X})$ ; *to provide the local semantics of the policy.*
- (iv)  $Q$  is a set of terms from  $\mathcal{T}(\mathcal{F})$ :  $Q \subseteq \mathcal{T}(\mathcal{F})$ ; *to describe the form of the acceptable requests.*
- (v)  $\zeta$  is a rewrite strategy for  $R$ ; *to guide the rule application and therefore provide the global semantics of the policy.*

The notions of *local* and *global* semantics mentioned in the comments of this defi-

dition are provided as an intuitive way to describe the fact that a rewrite rule is an entity that describes a local transformation and a strategy a way to combine these local application rules. This is formally developed in the next section (section 3).

In order to illustrate this definition, let us consider the following simple example.

**Example 2.2** This example is taken from the NetFilter `how-to`<sup>2</sup>. Suppose an Internet user wants to set his firewall to block any traffic coming from the exterior to his local network. Since the interface associated to Internet connections is usually `ppp0`, a simple method is to reject all new packets coming from this source. In order to demonstrate the fact that it might be convenient for a policy to contain rules beyond those which *directly* compute decisions, we also give some additional rules which allow two different local computers to share the same external IP address: for each outgoing packet whose origin is a local machine, its head is rewritten to a single address.

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

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

The function  $pkt$  computes a decision for the packets being transmitted in the network.

- The set of constant symbols representing decisions is  $D = \{accept, drop\}$ .
- Consider  $R$  as the following rules, where  $src, dst : Address$ , and  $s : State$  are variables:

$$\begin{aligned}
 pkt(src, dst, estab) &\rightarrow accept \\
 pkt(eth0, dst, new) &\rightarrow accept \\
 pkt(ppp0, dst, new) &\rightarrow drop \\
 pkt(10.1.1.1, ppp0, s) &\rightarrow pkt(123.123.1.1, ppp0, s) \\
 pkt(10.1.1.2, ppp0, s) &\rightarrow pkt(123.123.1.1, ppp0, s)
 \end{aligned}$$

The first rule says that packets concerning established connections have to be accepted. The second rule matches any packet whose origin is the local network, which are accepted. The third rule rejects any new packet coming from the external network (`ppp0`). The last two rules match the IP address of the local machines and rewrites them to a single IP address visible from the external network.

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

<sup>2</sup> <http://www.netfilter.org>

- A common strategy for such firewall policy, is to apply rules in the order they are given. We will see how to formalize this strategy in the following section.

The above elements define a security policy. It is worth noticing the presence of recursive rules, which are important for the policy definition, but that do not directly derive permissions.

Further examples to illustrate this definition are given for instance in [13,9,11]. In this framework, policy evaluation corresponds to evaluation by strategic rewriting of ground requests. Let us introduce this concept in the next section.

### 3 Strategic rewriting and narrowing

Strategic rewriting and narrowing are defined and studied in [26]; the interested reader can refer to this paper for more details.

Given a rewrite system  $R$ , a term  $t$  in  $\mathcal{T}(\mathcal{F}, \mathcal{X})$  rewrites to another term  $t'$  if there exists a rewrite rule  $l \rightarrow r$  of  $R$ , a position  $\omega$  in  $t$ , and a substitution  $\sigma$  such that  $t|_{\omega} = \sigma l$  and  $t' = t[\sigma r]_{\omega}$ . This is written  $t \rightarrow_{\omega, l \rightarrow r, \sigma}^R t'$  where either  $\omega$ ,  $l \rightarrow r$ ,  $\sigma$  or  $R$  may be omitted.  $t|_{\omega}$  is called a *redex*. A term that has no redex is said to be irreducible for  $R$  or to be in  $R$ -normal form, or simply normalized. The reflexive transitive closure of the rewriting relation induced by  $R$  is denoted by  $\rightarrow^* R$ .

A term rewrite system  $R$  generates an abstract reduction system, i.e. a labeled oriented graph  $\mathcal{R} = (\mathcal{O}_R, \mathcal{S}_R)$  whose nodes are terms  $\mathcal{O}_R \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$ , and whose oriented edges are rewriting steps:  $\mathcal{S}_R = \{t \rightarrow t' \mid t \rightarrow_{\omega}^R t' \text{ for } \omega \text{ a position in } t\}$ . An  $\mathcal{R}$ -*derivation* or  $\mathcal{R}$ -*reduction sequence* is a path  $\pi$  in the graph  $\mathcal{R}$ ; when it is finite,  $\pi$  can be written  $a_0 \rightarrow^{\phi_0} a_1 \rightarrow^{\phi_1} a_2 \dots \rightarrow^{\phi_{n-1}} a_n$  and we say that  $a_0$  reduces to  $a_n$  by the derivation  $\pi = \phi_0 \phi_1 \dots \phi_{n-1}$ ; this is also denoted  $a_0 \rightarrow^{\phi_0 \cdot \phi_1 \dots \phi_{n-1}} a_n$  or simply  $a_0 \rightarrow^{\pi} a_n$ ;  $n$  is the *length* of  $\pi$ . A derivation is *empty* when its length is zero, in which case its source and target are the same. The empty derivation issued from  $a$  is denoted  $id_a$ . The *source* of  $\pi$  is the singleton  $\{a_0\}$  denoted  $dom(\pi)$ . The *target* of  $\pi$  is the singleton  $\{a_n\}$  and it is denoted  $(\pi a_0)$  or simply  $\pi a_0$  when there is no syntactic ambiguity; note that an  $\mathcal{R}$ -derivation is the concatenation of its reduction steps. The concatenation of  $\pi_1$  and  $\pi_2$  when it exists, is a new  $\mathcal{R}$ -derivation.

It is common to identify a rewrite system (i.e., a set of rewrite rules) with the abstract reduction system it generates (i.e., the set of all derivations allowed by  $R$ ). To a rewrite system corresponds directly a unique abstract reduction system that can be seen as a generic way to describe the set of all derivations. The converse is not true since from a set of derivations, the generating rewrite system is not in general uniquely determined.

#### 3.1 Strategic Rewriting

**Definition 3.1** [Abstract Strategy] For a given abstract reduction system  $\mathcal{R}$ : An *abstract strategy*  $\zeta$  is a subset of the set of all derivations (finite or not) of  $\mathcal{R}$ . Applying the strategy  $\zeta$  on an object  $a$  is denoted  $\zeta a$ . It denotes the set of all objects that can be reached from  $a$  using a derivation in  $\zeta$ :

$$\zeta a = \{b \mid \exists \pi \in \zeta \text{ such that } a \rightarrow^{\pi} b\} = \{\pi a \mid \pi \in \zeta\}$$

When no derivation in  $\zeta$  has source  $a$ , we say that the strategy application on  $a$  fails. Applying the strategy  $\zeta$  on a set of objects consists in applying  $\zeta$  to each element  $a$  of the set. The result is the union of  $\zeta a$  for all  $a$  in the set of objects. The *domain* of a strategy is the set of objects that are source of a derivation in  $\zeta$ :  $\text{dom}(\zeta) = \bigcup_{\delta \in \zeta} \text{dom}(\delta)$ . The strategy that contains all the empty derivations is  $\text{Id} = \{id_a \mid a \in \mathcal{O}\}$ .

It is now possible to give the definition of strategic rewriting:

**Definition 3.2** [Strategic rewriting [26]] Let  $\mathcal{R} = (\mathcal{O}_R, \mathcal{S}_R)$  be the abstract reduction system generated by a rewrite system  $R$  and  $\zeta$  be a strategy of  $\mathcal{R}$ . A strategic rewriting derivation (or rewriting derivation under strategy  $\zeta$ ) is an element of  $\zeta$ . A strategic rewriting step under  $\zeta$  is a rewriting step  $t \rightarrow^R t'$  that occurs in a derivation of  $\zeta$ , which is denoted  $t \rightarrow^\zeta t'$ .

A strategy can be described by enumerating all its elements or more suitably by a *strategy language*. Various approaches have been followed, yielding slightly different strategy languages such as ELAN [28,7], Stratego [32], Tom [5,4]<sup>3</sup> or more recently Maude [8]. All these languages share the concern to provide abstract ways to express control of rule applications, by using reflexivity and the meta-level for Maude, or the notion of rewriting strategies for ELAN or Stratego. Strategies such as bottom-up, top-down or leftmost-innermost are higher-order features that describe how rewrite rules should be applied. Tom, ELAN and Stratego provide flexible and expressive strategy languages where high-level strategies are defined by combining low level primitives. We refer to [26] for more details on these strategy languages.

In the context of security policies that we want to consider here, a few strategies are of main interest: the strategy  $\text{universal}(R)$  represents all derivations generated by a set of rewrite rules  $R$ ,  $\text{innermost}(R)$  represents derivations where rules of  $R$  are applied only on terms whose all strict subterms are irreducible.  $\text{ordered}(r_1, \dots, r_k)$  represents derivations where rules  $r_1, \dots, r_k$  are tried in this order on terms whose all strict subterms are irreducible. This last strategy corresponds to Innermost Priority rewriting (IP-rewriting for short) that is defined below.

When a rewrite system  $R$  is partially ordered, we say that  $r_1$  has a higher priority than  $r_2$  if  $r_1$  is greater than  $r_2$ . Innermost Priority Rewriting can then be defined as follows:

**Definition 3.3** [Innermost Priority Rewriting [31]] Given a rewrite system  $R$  with a partial ordering on rules, a term  $t$  in  $\mathcal{T}(\mathcal{F}, \mathcal{X})$  IP-rewrites to  $t'$ , if there exists a rewrite rule  $l \rightarrow r$  of  $R$ , a position  $\omega$  in  $t$ , and a substitution  $\sigma$  such that  $t|_\omega = \sigma l$  and  $t' = t[\sigma r]_\omega$  such that  $t \xrightarrow{\omega, l \rightarrow r, \sigma}^R t'$ , no proper subterm of  $t|_\omega$  is IP-reducible, and  $t|_\omega$  is not reducible by any other rule of higher priority than  $l \rightarrow r$ . This is written  $t \xrightarrow{\omega, l \rightarrow r, \sigma}^{IP} t'$  where either  $\omega$ ,  $l \rightarrow r$ ,  $\sigma$  or  $R$  may be omitted.  $t|_\omega$  is called a redex. A term that has no redex is said to be IP-normalized.

A substitution is said (ground) IP-normalized if all the terms in its image are (ground) IP-normalized.

<sup>3</sup> <http://tom.loria.fr>

### 3.2 The narrowing process

The narrowing process, introduced in [24,15], is quite similar to rewriting but there, matching is replaced by unification. Let us recall its usual definition:

**Definition 3.4** [Narrowing] Given a rewrite system  $R$ , a term  $t$  in  $\mathcal{T}(\mathcal{F}, \mathcal{X})$  narrows to the term  $t'$  if there exists a rewrite rule  $l \rightarrow r$  of  $R$  and a position  $\omega$  in  $t$  and such that  $t|_\omega$  and  $l$  are unifiable with a most general unifier  $\sigma$ . Then  $t' = \sigma(t[r]_\omega)$ . This is denoted  $t \rightsquigarrow_{\omega, l \rightarrow r, \sigma}^R t'$  or  $t \rightsquigarrow t'$  when we do not need to make precise which rewrite rule is used and where.

Let us remember that the variables in a rewrite rule are implicitly universally quantified and therefore, the set of variables in a rewrite rule may always be assumed distinct from those in the narrowed term (i.e. in the definition above,  $\text{Var}(t) \cap \text{Var}(l) = \emptyset$ ). Let us remark also that narrowing subsumes rewriting since a match between variable disjoint terms is also a unifier.

Based now on the narrowing process, a term rewrite system  $R$  generates another abstract reduction system  $\mathcal{N} = (\mathcal{O}_R, \mathcal{S}_R)$  with  $\mathcal{O}_R \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$ , and  $\mathcal{S}_R = \{t \rightsquigarrow t' \mid t \rightsquigarrow_{\omega, l \rightarrow r}^R t' \text{ for } \omega \text{ a position in } t\}$ . As for the rewriting relation, we can define strategic narrowing.

**Definition 3.5** [Strategic narrowing [26]] Given an abstract reduction system  $\mathcal{N} = (\mathcal{O}_R, \mathcal{S}_R)$  generated by a rewrite system  $R$ , and a strategy  $\zeta$  of  $\mathcal{N}$ , a strategic narrowing derivation (or narrowing derivation under strategy  $\zeta$ ) is an element of  $\zeta$ . A strategic narrowing step under  $\zeta$  is a narrowing step  $t \rightsquigarrow^R t'$  that occurs in a derivation of  $\zeta$ , which is denoted  $t \rightsquigarrow^\zeta t'$ .

### 3.3 Simulation of rewriting by narrowing

We are now interested in formalizing precisely how (strategic) rewriting and (strategic) narrowing are related. This relation is the crucial property needed for queries solving. It may be expressed thanks to the general notion of simulation on abstract reduction systems given in [21] and also independently defined in [14].

**Definition 3.6** [Simulation] Let  $(\mathcal{O}_1, \rightarrow_1)$  and  $(\mathcal{O}_2, \rightarrow_2)$  be two abstract reduction systems.  $(\mathcal{O}_1, \rightarrow_1)$   $\Gamma$ -simulates  $(\mathcal{O}_2, \rightarrow_2)$  for a relation  $\Gamma \subseteq \mathcal{O}_1 \times \mathcal{O}_2$  when for every reduction step  $a_2 \rightarrow_2 b_2$  there is a corresponding reduction step  $a_1 \rightarrow_1 b_1$  such that  $a_1 \Gamma a_2$  and  $b_1 \Gamma b_2$ .

Let us define the relation  $\Gamma \subseteq \mathcal{T}(\mathcal{F}) \times \mathcal{T}(\mathcal{F}, \mathcal{Y})$  as  $t \Gamma u$  if  $t$  is a ground instance of  $u$ . The following lemma expresses the fact that the abstract reduction system  $(\mathcal{T}(\mathcal{F}), \rightarrow)$  simulates the abstract reduction system  $(\mathcal{T}(\mathcal{F}, \mathcal{Y}), \rightsquigarrow)$ .

**Lemma 3.7** For any term  $t \in \mathcal{T}(\mathcal{F}, \mathcal{Y})$  and rewrite system  $R$ , if  $t \rightsquigarrow_{\omega, l \rightarrow r, \sigma} t'$  then  $\sigma(t) \rightarrow_{\omega, l \rightarrow r, \sigma} \sigma(t')$ , and for any ground instance  $\alpha$  of  $\sigma(t)$ ,  $\alpha(\sigma(t)) \rightarrow_{\omega, l \rightarrow r, \sigma} \alpha(\sigma(t'))$ .

As a consequence, a derivation in the abstract reduction system  $(\mathcal{T}(\mathcal{F}, \mathcal{Y}), \rightsquigarrow)$  corresponds to a set of derivations in the abstract reduction system  $(\mathcal{T}(\mathcal{F}), \rightarrow)$ :

**Proposition 3.8** For any term  $t \in \mathcal{T}(\mathcal{F}, \mathcal{Y})$  and rewrite system  $R$ , if  $t \rightsquigarrow_{\omega_1, l_1 \rightarrow r_1, \sigma_1}^R t_1 \dots \rightsquigarrow_{\omega_n, l_n \rightarrow r_n, \sigma_n}^R t_n$ , then

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

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

$$\alpha(\sigma_n \dots \sigma_1(t)) \rightarrow_{\omega_1, l_1 \rightarrow r_1}^R \dots \rightarrow_{\omega_n, l_n \rightarrow r_n}^R \alpha(t_n).$$

Given a terminating rewrite system  $R$ , let us consider the subset  $\mathcal{N}$  of  $\mathcal{T}(\mathcal{F})$  of  $R$ -normalized ground instances of terms in  $\mathcal{T}(\mathcal{F}, \mathcal{X})$ . Let us now define the relation  $\Gamma \subseteq \mathcal{T}(\mathcal{F}, \mathcal{Y}) \times \mathcal{N}$  as  $: u\Gamma t$  if  $t$  is an  $R$ -normalized ground instance of  $u$ . The fact that the abstract reduction system  $(\mathcal{T}(\mathcal{F}, \mathcal{Y}), \rightsquigarrow)$  simulates the abstract reduction system  $(\mathcal{N}, \rightarrow)$  is more subtle to establish and is a consequence of the following lemma proved first by J.-M. Hullot [24].

**Lemma 3.9** Let  $t_0$  be a term and  $\rho$  be an  $R$ -normalized substitution such that  $\rho(t_0) \rightarrow_{\omega, l \rightarrow r}^R t'_1$ . Then there exist substitutions  $\sigma$  and  $\mu$  such that:

- (i)  $t_0 \rightsquigarrow_{\omega, l \rightarrow r, \sigma}^R t_1$ ,
- (ii)  $\mu(t_1) = t'_1$ ,  $\mu$  is  $R$ -normalized,
- (iii)  $\rho =_{\text{Var}(t_0)} \mu\sigma$  (i.e.  $\forall x \in \text{Var}(t_0), \rho(x) = \mu\sigma(x)$ ).

This result can be easily extended by induction on the number of steps to any rewriting derivation:

**Proposition 3.10** Let  $t_0$  be a term and  $\rho$  be an  $R$ -normalized substitution such that:

$$\rho(t_0) \rightarrow_{\omega_1, l_1 \rightarrow r_1}^R \dots \rightarrow_{\omega_n, l_n \rightarrow r_n}^R t'_n.$$

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

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

Narrowing derivations thus schematize rewriting derivations when strategies are not taken into account. In the more general case of strategic rewriting and narrowing, in order to obtain similar results, the two abstract reduction systems  $(\mathcal{T}(\mathcal{F}), \rightarrow)$  and  $(\mathcal{T}(\mathcal{F}, \mathcal{Y}), \rightsquigarrow)$  must simulate each other. Currently we only have simulation results for specific strategies such as innermost or outermost rewriting [21], or priority rewriting [19].

### 3.4 Simulation of IP-rewriting by IP-narrowing

Let us focus in this section on innermost priority rewriting and on the corresponding strategic narrowing relation defined in [19]. In order to mimic IP-rewriting on ground instances of terms, the narrowing process has to exclude some steps that do not correspond to any rewriting step in the strategy. So the substitutions used to narrow a term have in general to satisfy a set of disequalities coming from the

negation of non-relevant substitutions. To formalize this point, we first need the definition of constrained substitutions. In the following, we identify a substitution  $\sigma = (x_1 \mapsto t_1) \dots (x_n \mapsto t_n)$  on  $\mathcal{T}(\mathcal{F}, \mathcal{Y})$  with the finite set of solved equations  $(x_1 = t_1) \wedge \dots \wedge (x_n = t_n)$ , also denoted by the equality formula  $\bigwedge_i (x_i = t_i)$ , with  $x_i \in \mathcal{Y}$ ,  $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{Y})$ , where  $=$  is the syntactic equality. Similarly, we call *negation*  $\bar{\sigma}$  of the substitution  $\sigma$  the formula  $\bigvee_i (x_i \neq t_i)$ . A constrained substitution  $\sigma$  is then a formula  $\sigma_0 \wedge c_0$  where  $\sigma_0$  is a substitution and  $c_0$  a constraint  $\bigwedge_j \bigvee_{i_j} (x_{i_j} \neq t_{i_j})$ .

We can now give the definition of IP-narrowing:

**Definition 3.11** [IP-narrowing] [19] A term  $t \in \mathcal{T}(\mathcal{F}, \mathcal{Y})$  IP-narrows into a term  $t' \in \mathcal{T}(\mathcal{F}, \mathcal{Y})$  at the non-variable position  $\omega$  of  $t$ , using the rule  $l \rightarrow r \in R$  with the constrained substitution  $\sigma = \sigma_0 \wedge \bigwedge_{j \in [1..k]} \bar{\sigma}_j \wedge_{i \in [1..n]} \bar{\sigma}_0^i$ , which is written  $t \rightsquigarrow_{\omega, l \rightarrow r, \sigma}^{IP} t'$  iff  $\sigma_0(l) = \sigma_0(t|_\omega)$  and  $t' = \sigma_0(t[r]_p)$ , where  $\sigma_0$  is the most general unifier of  $t|_p$  and  $l$ ,  $\sigma_j, j \in [1..k]$  are all most general unifiers of  $\sigma_0 t|_{\omega'}$  and a left-hand side  $l'$  of a rule of  $R$ , for all suffix positions  $\omega'$  of  $\omega$  in  $t$ , and  $\sigma_0^1, \dots, \sigma_0^n$  are the most general unifiers of  $t|_\omega$  with the left-hand sides of rules having a greater priority than  $l \rightarrow r$ .

Taking into account the restrictions put by constraints on substitutions, we get the following result.

**Lemma 3.12** For any term  $t \in \mathcal{T}(\mathcal{F}, \mathcal{Y})$  and rewrite system  $R$  with priorities, if  $t \rightsquigarrow_{\omega, l \rightarrow r, \sigma \wedge c}^{IP} t'$  then for any ground IP-normalized instance  $\alpha$  of  $\sigma(t)$  that satisfies  $c$ ,  $\alpha(\sigma(t)) \rightarrow_{\omega, l \rightarrow r}^{IP} \alpha(t')$ .

Consequently, a derivation in the abstract reduction system  $(\mathcal{T}(\mathcal{F}, \mathcal{Y}), \rightsquigarrow^{IP})$  corresponds to a set of derivations in the abstract reduction system  $(\mathcal{T}(\mathcal{F}), \rightarrow^{IP})$ .

**Proposition 3.13** For any term  $t \in \mathcal{T}(\mathcal{F}, \mathcal{Y})$  and rewrite system  $R$  with priorities, if  $t \rightsquigarrow_{\omega_1, l_1 \rightarrow r_1, \sigma_1 \wedge c_1}^{IP} t_1 \dots \rightsquigarrow_{\omega_n, l_n \rightarrow r_n, \sigma_n \wedge c_n}^{IP} t_n$ , then for any ground IP-normalized instance  $\alpha$  of  $\sigma_n \dots \sigma_1(t)$  satisfying  $c_1 \wedge \dots \wedge c_n$ ,

$$\alpha(\sigma_n \dots \sigma_1(t)) \rightarrow_{\omega_1, l_1 \rightarrow r_1}^{IP} \dots \rightarrow_{\omega_n, l_n \rightarrow r_n}^{IP} \alpha(t_n).$$

Conversely the following lifting lemma is due to [19] and similar to those in [20].

**Lemma 3.14 (IP-lifting Lemma)** Let  $R$  be a rewrite system with priorities. Let  $s \in \mathcal{T}(\mathcal{F}, \mathcal{Y})$ ,  $\alpha$  a ground substitution such that  $\alpha s$  is IP-reducible at a non-variable position  $\omega$  of  $s$ , and  $\mathcal{Y}' \subseteq \mathcal{Y}$  a set of variables such that  $\text{Var}(s) \cup \text{Dom}(\alpha) \subseteq \mathcal{Y}'$ . If  $\alpha(s) \rightarrow_{\omega, l \rightarrow r}^{IP} t'$ , then there exist a term  $s' \in \mathcal{T}(\mathcal{F}, \mathcal{Y})$  and substitutions  $\beta, \sigma = \sigma_0 \wedge \bigwedge_{j \in [1..k]} \bar{\sigma}_j \wedge_{i \in [1..n]} \bar{\sigma}_0^i$  such that:

- (i)  $s \rightsquigarrow_{\omega, l \rightarrow r, \sigma}^{IP} s'$
- (ii)  $\beta s' = t'$
- (iii)  $\beta \sigma_0 = \alpha[\mathcal{Y}' \cup \text{Var}(l)]$
- (iv)  $\beta$  satisfies  $\bigwedge_{j \in [1..k]} \bar{\sigma}_j \wedge_{i \in [1..n]} \bar{\sigma}_0^i$

where  $\sigma_0$  is the most general unifier of  $s|_\omega$  and  $l$ , for  $j \in [1..k]$  the  $\sigma_j$  are the most general unifiers of  $\sigma_0 s|_{\omega'}$  with a left-hand side  $l'$  of a rule of  $R$ , for all suffix positions  $\omega'$  of  $\omega$  in  $s$ , and  $\sigma_0^1, \dots, \sigma_0^n$  are the most general unifiers of  $s|_\omega$  with the left-hand sides of the rules having a greater priority than  $l \rightarrow r$ .

If  $\alpha$  is IP-normalized,  $\beta$  is also IP-normalized. This result can be extended by induction on the number of steps to rewriting derivations:

**Proposition 3.15** *Let  $t_0$  be a term and  $\rho$  be an IP-normalized substitution such that  $\rho(t_0) \xrightarrow{\omega_1, l_1 \rightarrow r_1}^{IP} \dots \xrightarrow{\omega_n, l_n \rightarrow r_n}^{IP} t'_n$ . Then there exist IP-normalized constrained substitutions  $\sigma_i \wedge c_i (i = 1, \dots, n)$  and  $\mu$  such that:*

- (i)  $t_0 \rightsquigarrow_{\omega_1, l_1 \rightarrow r_1, \sigma_1 \wedge c_1}^{IP} t_1 \dots \rightsquigarrow_{\omega_n, l_n \rightarrow r_n, \sigma_n \wedge c_n}^{IP} t_n$ ,
- (ii)  $\mu(t_n) = t'_n$ ,
- (iii)  $\rho = \nu_{\text{var}(t_0)} \mu \sigma_n \dots \sigma_1$ ,
- (iv)  $\mu$  satisfies  $c_1 \wedge \dots \wedge c_n$ .

**Proof** (Idea) Since  $\rho$  is IP-normalized, the reduction occurs in  $t_0$  which is IP-narrowable according to Lemma 3.14. But then  $\beta$  is IP-normalized. The proof goes on as in Proposition 3.10.  $\square$

## 4 Narrowing-based analysis of rewriting-based policies

In our framework, policy evaluation is performed by strategic rewriting of ground requests. Consequently, for the safe design of security policies, several properties have been identified [13,10].

**Definition 4.1** [Termination, Consistency, Decision Completeness] [10] 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.
- *consistent* if for every query  $q \in Q$ ,  $\zeta$  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.
- *decision complete* if  $\forall q \in Q, \exists d \in D$ , such that  $d \in \zeta q$ .

To deal with policy analysis, we add now the possibility to express queries:

**Definition 4.2** [Queries] 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  $\mathcal{X}$ .

Based on the previous results of Section 3, we can now analyze the behavior of security policies by solving queries. 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. The general intuitive idea is that the narrowing trees provide a good simulation of the rewriting derivation trees that correspond to requests evaluation of the policy. We make the assumption that the security policy is terminating, in order to consider normalized terms, substitutions and instantiations. There are quite powerful techniques to check termination such as recursive path orderings [12], semantic labeling [33], dependency pairs [2], etc. These and other techniques have been implemented by several tools that allow to verify termination for a large sets of rewrite systems: for example AProVE [18], TTT [23], and CiME [30], to cite a few. All these verification methods and tools check termination statically, which is very attractive for the specification of flexible policies.

Considering innermost termination in particular, specific methods have been given for instance in [1,17,20]. Techniques have also been studied for termination of outermost rewriting [20], for termination of lazy rewriting [16] or for termination of priority rewriting [31,19]. Moreover it may sometimes be easier to prove termination in general (i.e. for the **universal** strategy), which implies termination under any strategy.

**Example 4.3** The policy from Example 2.2 is easily proved terminating. This can be easily checked by analyzing the rules of the policy and giving a simplification ordering  $>$  such that  $l > r$  for any left- and right-hand sides  $l, r$  of rules in  $R$ . This guarantees that the evaluation of any request will always terminate on a resulting term.

#### 4.1 Assumptions on the policies and definitions

For simplicity, we restrict in this section to policies with universal strategies. Thanks to the theoretical results of Section 3 stated both for rewriting and IP-rewriting, we conjecture that the narrowing-based analysis can be performed for IP-rewriting strategies as well as for universal ones.

**Definition 4.4** Given a security policy  $\wp = (\mathcal{F}, D, R, Q, \zeta)$ , let  $Def$  be the subset of  $\mathcal{F}$  composed of all symbols at top positions of left-hand sides of rules.  $\wp$  is *simple* if it is terminating, the decisions in  $D$  are irreducible and the requests  $Q$  are ground terms whose top symbol is in  $Def$ .

For instance, the policy for firewall of Example 2.2 is simple. There  $D = \{\text{accept}, \text{drop}\}$  and  $Def = \{\text{pkt}\}$ . The policy is terminating since  $R$  universally terminates.

#### 4.2 What-if analysis

Let us first consider queries of the form  $q(x_1, \dots, x_n)$  where  $q$  is a term in  $\mathcal{T}(\mathcal{F}, \mathcal{Y})$  with variables  $x_1, \dots, x_n \in \mathcal{Y}$ .

For a term  $u$  with variables, we denote by  $\langle u \rangle$  the set  $\{\theta(u) \mid \theta \in \Phi\}$  where  $\Phi$  is the set of all ground normalized instances of  $\mathcal{T}(\mathcal{F}, \mathcal{Y})$  into  $\mathcal{T}(\mathcal{F})$ .

**Proposition 4.5** For a simple security policy  $\wp$ , let us consider the narrowing tree of the query  $q(x_1, \dots, x_n)$ .

- (i) For every node  $u$  such that  $q(x_1, \dots, x_n) \rightsquigarrow_{\sigma}^* u$ , all requests  $\langle \sigma(q(x_1, \dots, x_n)) \rangle$  evaluate to requests in  $\langle u \rangle$ , where  $\sigma$  is the composition of normalized narrowing substitutions used in the narrowing derivation.
- (ii) For any request  $t$  reachable by the policy from a request  $\alpha(q(x_1, \dots, x_n))$ , there exists a node  $u$  in the narrowing tree of  $q(x_1, \dots, x_n)$ , such that  $q(x_1, \dots, x_n) \rightsquigarrow_{\sigma}^* u$  and  $t = \mu(u)$ , with  $\mu$  normalized if  $\alpha$  is normalized.

#### Proof

- (i) is a direct application of Proposition 3.8.
- (ii) directly follows from Proposition 3.10.

□

Based on this result, we can make some what-if analysis, as shown on the simple firewall Example 2.2.

**Example 4.6** Suppose we want to know which packets get accepted for a new connection. This amounts to solve the query  $pckt(x, y, new)$ . We get the following narrowing substitutions with the respective rules:

- (1)  $x = eth0 \quad y = dst$
- (2)  $x = ppp0 \quad y = dst$
- (3)  $x = 10.1.1.1 \quad y = ppp0 \quad new = s$
- (4)  $x = 10.1.1.2 \quad y = ppp0 \quad new = s$

giving respectively *accept*, *drop* and  $pckt(123.123.1.1, ppp0, new)$  twice. These terms are no more narrowable. So possible requests that give an *accept* decision are ground instances of  $pckt(eth0, dst, new)$ .

### 4.3 Analyzing decisions

Another interest of the approach is to provide also a methodology for analyzing the decisions taken by the policy. Let us first state some terminology and definitions.

A *decision term* is a ground term of  $D$ . A *query pattern* is a term of the form  $f(x_1, \dots, x_n)$  with  $f \in Def$  and  $x_1, \dots, x_n \in \mathcal{Y}$ . It will schematize in the following the set of all ground normalized instances  $\langle f(x_1, \dots, x_n) \rangle$ . In order for the query patterns to represent the set of all possible requests of the policy, we need to provide an appropriate notion of completeness:

**Definition 4.7** A simple security policy is *completely defined* if for every query pattern  $f(x_1, \dots, x_n)$  with  $f \in Def$ , all ground normalized instances  $\theta(f(x_1, \dots, x_n))$  are reducible.

Then we can state the following results.

**Proposition 4.8** For a simple security policy  $\wp$ , let us consider the narrowing trees of the query patterns  $f(x_1, \dots, x_n)$ . If  $f(x_1, \dots, x_n) \rightsquigarrow_{\sigma}^* u$  and  $u$  is a decision term, where  $\sigma$  is the composition of normalized narrowing substitutions, then ground normalized instances of  $\sigma(f(x_1, \dots, x_n))$  give a set of requests leading to the decision  $u$ .

**Proof** This is a direct application of Proposition 3.8. □

Provided decisions are normalized terms (which is trivial when decisions are constants on which no rewrite rule applies), for a simple security policy  $\wp$  completely defined, in order to generate all possible requests leading to a given decision, we can collect all the narrowing branches leading to this decision (obviously a leaf since a decision is no more reducible nor narrowable) and consider the union of the composed substitutions along each branch.

One can also prove that a given pattern is not reachable, by trying to unify this pattern with each node. If no node unifies with the pattern, the latter is unreachable. In our framework, this gives the following result:

**Proposition 4.9** *If a decision  $d$  is not an instance of any node in the narrowing tree of  $f(x_1, \dots, x_n)$ , there is no request in  $\langle f(x_1, \dots, x_n) \rangle$  leading to  $d$ .*

**Proof** If it were the case, there would exist  $\alpha(f(x_1, \dots, x_n)) \xrightarrow{*} d$ , with  $\alpha$  ground normalized, and then a node  $u$  such that  $f(x_1, \dots, x_n) \rightsquigarrow_{\sigma}^* u$  and  $d = \mu(u)$ , which contradicts the hypothesis.  $\square$

Let us come back to Example 2.2.

**Example 4.10** Suppose we want to know which packets get accepted and which are dropped. This amounts to solve the query  $pckt(x, y, z)$ . We get the following narrowing substitutions with the respective rules:

- (1)  $x = src \quad y = dst \quad z = estab$
- (2)  $x = eth0 \quad y = dst \quad z = new$
- (3)  $x = ppp0 \quad y = dst \quad z = new$
- (4)  $x = 10.1.1.1 \quad y = ppp0 \quad z = s$
- (5)  $x = 10.1.1.2 \quad y = ppp0 \quad z = s$

Narrowing with the two first rules transforms the query  $pckt(x, y, z)$  into *accept*, while the third rule transforms the query into *drop*. Narrowing with the two last rules transforms the query  $pckt(x, y, z)$  into  $pckt(123.123.1.1, ppp0, s)$ . Then in each case a narrowing step can be again applied with the first rule, getting *accept* with  $z = estab$ . The two composed substitutions are (6)( $x = 10.1.1.1, y = ppp0, z = estab$ ) and (7)( $x = 10.1.1.2, y = ppp0, z = estab$ ).

However the instance of the query  $pckt(123.123.1.1, ppp0, new)$  is not reducible in this system. If we add to the system the rule  $pckt(123.123.1.1, ppp0, new) \rightarrow accept$ , the policy becomes completely defined. There is then one more branch in the narrowing tree leading to *accept* with the substitution (8)( $x = 123.123.1.1, y = ppp0, z = new$ ). A complete set of queries leading to the decision *accept* is given by the set of substitutions  $\{(1), (2), (6), (7), (8)\}$ .

## 5 Conclusion

The general context of this research is the design of a foundational framework for the compositional design, maintenance and verification of security policies. The framework initially presented in [13,11] addresses the problem of authoring and analyzing access control policies in a modular way using techniques developed in the field of strategic rewriting. Rewrite rules transform input terms representing access requests into access decision terms. In order to tame the raw computational power of term rewriting and to enhance the agility of the policy specification language, strategies are used to explicitly control the rule application. In this approach, the

correspondences between the properties of the rewrite system and the policy it implements are easy to understand, thus we are able to directly apply a rich corpus of existing proof techniques and tools. We have proposed here to use narrowing to contribute to improve the trust of a user or an administrator with respect to a policy, not only at the design stage, but also whenever a policy is updated. However, adapting new concepts such as strategic rewriting or strategic narrowing is yet an ongoing goal which this paper begins to contribute to. Future work concerns fine-tuned analysis tools for policies, especially to formalize and tackle the various strategies used by developers of policies.

## References

- [1] T. Arts and J. Giesl. Proving innermost normalisation automatically. In *Proceedings 8th Conference on Rewriting Techniques and Applications, Sitges (Spain)*, volume 1232 of *Lecture Notes in Computer Science*, pages 157–171. Springer-Verlag, 1997.
- [2] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
- [3] F. Baader and T. Nipkow. *Term Rewriting and all That*. Cambridge University Press, 1998.
- [4] E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles. **Tom Manual**. LORIA, Nancy (France), version 2.4 edition, October 2006.
- [5] E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles. Tom: Piggybacking rewriting on java. In F. Baader, editor, *RTA*, volume 4533 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2007.
- [6] S. Barker and P. J. Stuckey. Flexible access control policy specification with constraint logic programming. *ACM Trans. Inf. Syst. Secur.*, 6(4):501–546, 2003.
- [7] P. Borovanský, C. Kirchner, H. Kirchner, and C. Ringeissen. Rewriting with strategies in ELAN: a functional semantics. *International Journal of Foundations of Computer Science*, 12(1):69–98, February 2001.
- [8] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. Reflection, metalevel computation, and strategies. In M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors, *All About Maude*, volume 4350 of *Lecture Notes in Computer Science*, pages 419–458. Springer, 2007.
- [9] A. S. de Oliveira. Rewriting-based access control policies. *Electr. Notes Theor. Comput. Sci.*, 171(4):59–72, 2007.
- [10] A. S. de Oliveira. *Réécriture et modularité pour les politiques de sécurité*. PhD thesis, UHP Nancy 1, 2008.
- [11] A. S. de Oliveira, E. K. Wang, C. Kirchner, and H. Kirchner. Weaving rewrite-based access control policies. In *FMSE '07: Proceedings of the 2007 ACM workshop on Formal methods in security engineering*, pages 71–80, New York, NY, USA, 2007. ACM.
- [12] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1 & 2):69–116, 1987.
- [13] D. J. Dougherty, C. Kirchner, H. Kirchner, and A. S. de Oliveira. Modular access control via strategic rewriting. In *Proceedings of 12th European Symposium On Research In Computer Security (ESORICS'07)*, pages 578–593, Dresden, Sep 2007.
- [14] S. Escobar and J. Meseguer. Symbolic model checking of infinite-state systems using narrowing. In F. Baader, editor, *Term Rewriting and Applications. 18th International Conference, RTA 2007*, volume 4533 of *lncs*, 2007.
- [15] M. Fay. First order unification in equational theories. In *Proceedings 4th Workshop on Automated Deduction, Austin (Tex., USA)*, pages 161–167, 1979.
- [16] J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated Termination Analysis for Haskell: From term rewriting to programming languages. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications*, pages 297–312, 2006.
- [17] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2850 of *Lecture Notes in Artificial Intelligence*, pages 165–179, Almaty, Kazakhstan, 2003. Springer-Verlag.

- [18] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with *aprove*. In V. van Oostrom, editor, *RTA*, volume 3091 of *Lecture Notes in Computer Science*, pages 210–220. Springer, 2004.
- [19] I. Gnaedig. Termination of priority rewriting. HAL-INRIA Open Archive Number inria-00243131, 2008.
- [20] I. Gnaedig and K. H. Termination of rewriting under strategies. *ACM Transactions on Computational Logic*, 2008. Accepted. Also as HAL-INRIA Open Archive Number inria-00113156.
- [21] I. Gnaedig and H. Kirchner. Narrowing, abstraction and constraints for proving properties of reduction relations. In H. Comon-Lundh, C. Kirchner, and H. Kirchner, editors, *Rewriting, Computation and Proof. Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of His 60th Birthday*, volume 4600 of *Lecture Notes in Computer Science*, pages 44–67. sv, 2007.
- [22] M. G. Gouda and A. X. Liu. Structured firewall design. *Computer Networks*, 51(4):1106–1120, 2007.
- [23] N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Inf. Comput.*, 205(4):474–511, 2007.
- [24] J.-M. Hullot. Canonical forms and unification. In W. Bibel and R. Kowalski, editors, *Proceedings 5th International Conference on Automated Deduction, Les Arcs (France)*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer-Verlag, July 1980.
- [25] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, 2001.
- [26] C. Kirchner, F. Kirchner, and H. Kirchner. Strategic computations and deductions. In *Festschrift in honor of Peter Andrews*, Studies in logic and the foundations of mathematics. Elsevier, 2008.
- [27] C. Kirchner and H. Kirchner. Rewriting, solving, proving. A preliminary version of a book available at [www.loria.fr/~ckirchne/rsp.ps.gz](http://www.loria.fr/~ckirchne/rsp.ps.gz), 1999.
- [28] C. Kirchner, H. Kirchner, and M. Vittek. Designing constraint logic programming languages using computational systems. In P. Van Hentenryck and V. Saraswat, editors, *Principles and Practice of Constraint Programming. The Newport Papers.*, chapter 8, pages 131–158. The MIT press, 1995.
- [29] J. W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter 6. Oxford University Press, 1990.
- [30] C. Marché and X. Urbain. Modular and incremental proofs of ac-termination. *J. Symb. Comput.*, 38(1):873–897, 2004.
- [31] C. K. Mohan. Priority rewriting: Semantics, confluence, and conditionals. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 278–291. Springer-Verlag, April 1989.
- [32] E. Visser. Stratego: A language for program transformation based on rewriting strategies. System description of Stratego 0.5. In A. Middeldorp, editor, *Rewriting Techniques and Applications (RTA'01)*, volume 2051 of *Lecture Notes in Computer Science*, pages 357–361. Springer-Verlag, May 2001.
- [33] H. Zantema. Termination of term rewriting by semantic labelling. *Fundam. Inform.*, 24(1/2):89–105, 1995.