Access Control with Non-deterministic and Probabilistic Attribute Retrieval

Jason Crampton[1], Charles Morisset[2], and Nicola Zannone[3]

[1] Royal Holloway, University of London
Jason.Crampton@rhul.ac.uk
[2] Newcastle University
Charles.Morisset@ncl.ac.uk
[3] Eindhoven University of Technology
n.zannone@tue.nl

Abstract. Attribute Based Access Control (ABAC) is becoming the reference model for the specification and evaluation of access control policies. In ABAC policies and access requests are defined in terms of pairs attribute names/values. The applicability of an ABAC policy to a request is determined by matching the attributes in the request with the attributes in the policy. Some languages supporting ABAC, such as PTaCL or XACML 3.0, take into account the possibility that some attributes values might not be correctly retrieved when the request is evaluated, and use complex decisions, usually describing all possible evaluation outcomes, to account for missing attributes. In this paper, we argue that the problem of missing attributes in ABAC can be seen as a non-deterministic attribute retrieval process, and we show that the current evaluation mechanism in PTaCL or XACML can return a complex decision that does not necessarily match with the actual possible outcomes. This, however, is problematic for the enforcing mechanism, which needs to resolve the complex decision into a conclusive one. We propose a new evaluation mechanism, explicitly based on non-deterministic attribute retrieval for a given request. We extend this mechanism to probabilistic attribute retrieval and implement a probabilistic policy evaluation mechanism for PTaCL in PRISM, a probabilistic model-checker.

1 Introduction

In order to introduce our research problem, consider the following situation. Alice is a security guard controlling the access to a building. If Bob wants to enter the building, he gives Alice some pieces of information, such as his name, his purpose of visit, etc. Alice submits this information to a security system, which should return a simple decision, such as "Bob is permitted access to the building" or "Bob is denied access to the building". Based on this decision, Alice can open the door for Bob or not. However, in some cases, the security system might return an *inconclusive* decision such as "Bob might be able to access the building, but it is not entirely sure" or "Bob is either permitted or denied access to the building". In such cases, Alice still needs to make a decision about Bob.

The general problem we investigate is to know whether Alice should rely on the inconclusive decision returned by the security system to make her final decision. In particular, we focus on the case where inconclusive decisions are sets of possible decisions, and can be triggered due to missing information, i.e., information about Bob or the environment that was not provided. We will show that in the case we are considering, the set of possible decisions returned by the security system is not always meaningful, and therefore Alice should not rely on it, which raises the higher question of why such a set is returned in the first place. We also show that, under some conditions, even if the decision returned by the security system is conclusive, Alice should not necessarily rely on it.

Following standard terminology, hereafter we refer to the security system described above as a *Policy Decision Point* (PDP), i.e., the point in the system in charge of making a decision based on an *access control policy*, and to Alice both as a *Policy Enforcement Point* (PEP), i.e., the point in charge of enforcing the decision made by the PDP, and as a *Decision Resolving Point*, i.e., the point in charge of transforming the decision from the PDP into a decision understandable by the PEP. The information about Bob that Alice submits to the PDP is called the *access request*.

An access request traditionally contains a subject, representing the entity performing the request, an object, representing the entity concerned by the request, and an access mode, representing the type of access performed. For instance, an access request in a filesystem can consist of a user asking to access a file in read, write or execute mode. A seminal way to define a PDP is with an access matrix [14], which is an extensional definition of all allowed accesses, such that each column and each row corresponds to a subject and an object, respectively, and the cell content represents the access mode granted for the corresponding subject over the corresponding object. The decision returned by the PDP is thus always conclusive: either the access mode is in the matrix or not.

Subsequent access control models have introduced different security concepts, to help define more complex access control policies. For instance, the Bell-LaPadula model [1] introduces the notion of lattice of security levels, RBAC [8] introduces the notion of user role, the Chinese-Wall model [3] introduces the notion of class of conflict of interest, etc. Many other concepts have been proposed in the literature, usually to address a specific concern. In particular, recent approaches propose a general attribute-based access control (ABAC) model, e.g., [21, 18, 6], where, roughly speaking, everything is an attribute, and an access request is simply a set of pairs of attribute name and value.

Although ABAC is quite expressive, since most security concepts can be described as attributes, it also raises the issue of attribute management: what happens if some attribute values are missing when evaluating the policy? Some policy languages take this aspect into account, at least partially. For instance, in XACML [18] and PTaCL [6] (which we describe in more detail in Section 2.2), when a policy tries to check whether an attribute matches a specific values and if this attribute is missing from the request, then the policy can behave as if the attribute both matches and does not, at the same time. This mechanism leads

a policy to return multiple decisions, either through explicit sets of decisions in PTaCL or through extended Indeterminate decisions in XACML 3.0[4].

Complex policy decisions can be quite useful, since they can indicate problems that happened during policy evaluation. However, they also need to be resolved to *conclusive* decisions, i.e., decisions understandable by the PEP. This process is for instance done by the resolution function in [5], which takes a set of decisions and returns a single decision, and these functions are further studied in [16]. In this paper, we therefore propose the following contributions:

- We first show that the decision sets returned by a PTaCL/XACML policy do not necessarily correspond with an intuitive interpretation of what those decisions mean.
- We then propose a declarative evaluation mechanism for PTaCL which matches this intuition. This mechanism is based on a *non-deterministic* evaluation [22], which simulates the non-determinism of retrieving the attributes forming the request: if a value for an attribute is missing from a request, we do not know whether it should be in it or not[5].
- Finally, we extend this non-deterministic evaluation to a probabilistic one, and we show how they can be mixed.
- The concepts presented in this paper are supported by an automatic translation of PTaCL policies into PRISM, which is a probabilistic model-checker.

Related work. Our work relies clearly on the PTaCL language [6] together with the definition of the ATRAP tool [10], which automatically analyses the safety of PTaCL policies. The notion of reducing complex decisions to simple, conclusive ones is also addressed in recent work [16], which focuses on decisions and operators, whereas we focus here on attribute retrieval. In terms of methodology, our approach follows that of Tschantz and Krishnamurthi [22], since we first establish some requirement for an access control evaluation mechanism, and we then analyse an existing language against those requirements.

Model-checking has been used in the past for access control, for instance Zhang et al. [25] propose a tool checking whether a particular goal can be reached within an access control policy; Fistler et al. [9] defined the tool Margrave, which can analyse role-based access-control policies; more recently, Ranise et al. [19, 20] have used model checking to analyse the safety problem with administrative policies. In this work, we mostly focus on the attribute retrieval problem rather than on the policy evaluation/analysis problem (although they are quite related). To the best of our knowledge, we are the first to investigate probabilistic attribute retrieval in access control.

[4] XACML 3.0 introduces an extended Indeterminate set to deal with error occurring during policy evaluation: Indeterminate(P), Indeterminate(D) and Indeterminate(PD). Intuitively, these decisions record the decision(s) that would have returned if no error had occurred. As noted in [16], the Indeterminate(P) effectively corresponds to the set of decisions {Permit, Not-Applicable}, and similarly for Indeterminate(D) and Indeterminate(PD).

[5] We assume non-forgeability of attribute values: if a value for an attribute belongs to a request, then it is genuine.

Table 1: Binary and unary operators on the target decision set $\{1, 0, \bot\}$

d_1	d_2	$\neg d_1$	$\sim d_1$	$d_1 \tilde{\sqcap} d_2$	$d_1 \sqcap d_2$	$d_1 \triangle d_2$	$d_1 \tilde{\sqcup} d_2$	$d_1 \sqcup d_2$	$d_1 \triangledown d_2$
1	1	0	1	1	1	1	1	1	1
1	0	0	1	0	0	0	1	1	1
1	$\bot$	0	1	$\bot$	$\bot$	1	1	$\bot$	1
0	1	1	0	0	0	0	1	1	1
0	0	1	0	0	0	0	0	0	0
0	$\bot$	1	0	0	$\bot$	0	$\bot$	$\bot$	0
$\bot$	1	$\bot$	0	$\bot$	$\bot$	1	1	$\bot$	1
$\bot$	0	$\bot$	0	0	$\bot$	0	$\bot$	$\bot$	0
$\bot$	$\bot$	$\bot$	0	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

2 Background: PTaCL

In this section, we present the language PTaCL [6], after a brief introduction to 3-valued logic, to establish the notation used throughout the paper. All definitions and examples in this section are taken from [6, 10].

2.1 3-valued Logic

The truth values in Boolean logic are 0 and 1, where 1 represents *true* and 0 represents *false*; 3-valued logic extends it by considering an additional value $\bot$ [12]. There can be multiple interpretations of this extra symbol, for instance, the *weak conjunction* and *weak disjunction* operators, defined in Fig. 1 by $\sqcap$ and $\sqcup$ respectively, consider $\bot$ as absorbing; on the other hand, the *strong conjunction* and *strong disjunction* operators, defined by $\tilde{\sqcap}$ and $\tilde{\sqcup}$ respectively, consider $\bot$ as being either 1 or 0, and therefore try to "resolve" $\bot$ as much as possible. The operators $\triangle$ and $\triangledown$ assume that $\bot$ is overridden by any other decision. The difference between these two operators is that $\triangle$ defines the precedence of 0 over 1 while $\triangledown$ defines the precedence of 1 over 0. It is worth noting that $\triangle$ and $\triangledown$ correspond to XACML 3.0 *deny-overrides* and *permit-overrides* operators respectively. Finally, $\neg$ is the negation operator, and $\sim$ is a "weakening" operator, transforming $\bot$ into 0. We refer the interested reader to [7] for further work on 3-valued logic in access control, in particular concerning monotonicity and functional completeness results.

2.2 PTaCL

PTaCL is attribute-based, which means that a request is a set of attribute name-value pairs $\{(n_1, v_1), \ldots, (n_k, v_k)\}$. For instance, the request $\{(\mathbf{nat}, \mathsf{FR})\}$ represents a request made by a French national. In addition, PTaCL uses policy targets [2, 4, 5, 17, 18, 24], which specify to which requests the policy is applicable. A target can either be:

– an *atomic target* is a pair (n, v), where n is an attribute name and v is an attribute value;
– a *composite target* of the form $\mathsf{op}(t_1, \ldots, t_n)$, where op represents an n-ary 3-valued logical operator. For the sake of simplicity, we focus here on the unary and binary operators defined in Fig. 1.

Given a request, a target evaluates to a single value in $\{1, 0, \bot\}$, intuitively indicating if the request matches, does not match, or does not contain the attributes required to evaluate its applicability, respectively. More formally, the semantics of an atomic target (n, v) for a request $q = \{(n_1, v_1), \cdots, (n_k, v_k)\}$ is given as:

$$[\![(n, v)]\!]_{\mathrm{P}}(q) = \begin{cases} 1 & \text{if } (n, v') \in q \text{ and } v = v', \\ \bot & \text{if } (n, v') \notin q, \\ 0 & \text{otherwise.} \end{cases}$$

Composite targets are inductively evaluated by applying the operator to the result of the evaluation of the sub-targets. In particular, note that the unary operator $\sim$ effectively behaves as indicating that the absence of attributes should be considered as if the attribute does not match the value[6].

An *authorisation policy* can be:

– a *single decision*, i.e., either 1 (allow) or 0 (deny);
– a *targeted policy* (t, p), where t is a target and p is an authorisation policy;
– a *composite policy* $\mathsf{op}(p_1, \ldots, p_n)$, where op is an n-ary operator and $p_1, \ldots, p_n$ are authorisation policies. Here again, we focus on the operators defined in Fig. 1.

The evaluation of a single decision p for a request q is trivial:

$$[\![p]\!]_{\mathrm{P}}(q) = \begin{cases} 1 & \text{if } p = 1, \\ 0 & \text{if } p = 0. \end{cases}$$

The evaluation of a targeted policy requires considering whether the target of the policy matches the request. In general, for a given request, an attribute can be completely missing (for instance, in some cultures, people do not have a surname, and therefore an access request concerning such a person would have no value for the attribute corresponding to the surname), have exactly one value, or have multiple values (for instance with dual citizenships). However, because it is not necessarily known in advance the number of values an attribute can take for a particular request, it is impossible to know whether some values have been removed or not. PTaCL handles such situations by considering that if the target of a policy evaluates to $\bot$, then the policy must evaluate as if the target

[6] In other words, PTaCL considers that all attributes must be present by default, in XACML terminology, but the "indeterminate" value $\bot$ can always be transformed into the non-match value 0 using the $\sim$ operator.

Table 2: p_1 evaluation in PTaCL

	t_1	p_1	t_2	p_2
$\emptyset$	$\bot$	$\{1,0\}$	$\bot$	$\{1,0\}$
$\{(\mathbf{nat},\mathsf{FR})\}$	0	$\{1\}$	1	$\{1\}$
$\{(\mathbf{nat},\mathsf{AT})\}$	1	$\{0\}$	0	$\{0\}$
$\{(\mathbf{nat},\mathsf{FR}),(\mathbf{nat},\mathsf{AT})\}$	1	$\{0\}$	1	$\{1\}$

evaluates to both 1 and 0. More formally, the evaluation of a targeted policy (t,p) for a request q is given by:

$$[\![(t,p)]\!]_{\mathrm{P}}(q) = \begin{cases} [\![p]\!]_{\mathrm{P}}(q) & \text{if } [\![t]\!]_{\mathrm{P}}(q) = 1, \\ \{\bot\} & \text{if } [\![t]\!]_{\mathrm{P}}(q) = 0, \\ \{\bot\} \cup [\![p]\!]_{\mathrm{P}}(q) & \text{otherwise.} \end{cases}$$

where $\bot$ represents the not-applicable decision. Composite policies $\mathsf{op}(p_1,\ldots,p_n)$ are inductively evaluated by applying the operator to the result of the evaluation of the sub-policies. Conflicts that can arise from the evaluation of sub-policies are resolved based on the semantics of the combining operator op as defined in Fig. 1. It is worth emphasising that even though the evaluation of both targets and policies uses the set $\{1,0,\bot\}$, these values have a different interpretation in each case: they stand for "match", "non-match" and "indeterminate" when evaluating targets, and for "allow", "deny" and "not-applicable" when evaluating policies.

Let consider the following two policies: the policy $p_1 = \neg \sim \neg((\mathbf{nat},\mathsf{AT}),0)$ explicitly denies any access to Austrian citizens and otherwise allows the access; similarly, the policy $p_2 = \sim((\mathbf{nat},\mathsf{FR}),1)$ explicitly authorises any access to French citizens and otherwise denies the access. The evaluation of p_1 and p_2 for four different requests is given in Table 2. As shown in the table, the evaluation of the empty request against both p_1 and p_2 returns $\{1,0\}$. This result for p_1 can be derived as follows (the steps for the evaluation of the empty request against p_2 are similar):

$$[\![(\mathbf{nat},\mathsf{AT})]\!]_{\mathrm{P}}(\emptyset) = \bot$$

(Recall that in PTaCL, by default, a missing attribute is treated as an error in the target evaluation.)

$$\begin{aligned} [\![(\mathbf{nat},\mathsf{AT}),0)]\!]_{\mathrm{P}}(\emptyset) &= \{0,\bot\} \\ [\![\neg(\mathbf{nat},\mathsf{AT}),0)]\!]_{\mathrm{P}}(\emptyset) &= \{1,\bot\} \\ [\![\sim\neg((\mathbf{nat},\mathsf{AT}),0)]\!]_{\mathrm{P}}(\emptyset) &= \{1,0\} \\ [\![\neg\sim\neg((\mathbf{nat},\mathsf{AT}),0)]\!]_{\mathrm{P}}(\emptyset) &= \{0,1\} \end{aligned}$$

The other requests, on the other hand, present the attribute required by the target of p_1 and p_2, although their value may be different from the one in

the target. Thus, differently from the empty request, these requests evaluate to "match" or "non-match" depending on whether the attribute value in the request is the same of the one in the target of the request. For instance, request $(\mathbf{nat}, \mathsf{AT})$ is a "match" with respect to the target of p_1 and a "non-match" with respect to the target of p_2. Then, policy evaluation is performed as illustrated above.

3 Access Control with Incomplete Requests

As said in the previous section, ABAC introduces the possibility that some attributes might be missing from a request. Moreover, missing attributes cannot always be automatically detected. PTaCL addresses this possibility by letting targets evaluate to $\perp$ when an attribute required by the target is missing, which, in turn, can cause a policy to return multiple values. In this section, we first introduce some general characterisation of requests, after which we establish some requirements for policy evaluation mechanisms, and then show that PTaCL does not necessarily meet them. We then propose an abstract mechanism satisfying them, and we show in the following section how to implement it using the model-checker PRISM.

3.1 Request Characterisation

The major issue with missing information is that it is not necessarily syntactically detectable. Indeed, some attributes might simply not exist in a particular context, or the number of values expected for a given attribute is not necessarily known in advance. For instance, the request $\{(\mathbf{nat}, \mathsf{FR})\}$ has all required information if the subject submitting it has only the French nationality, but is missing information if she has multiple nationalities. Similarly, the empty request is missing information if the subject has a nationality, but is not if the subject is stateless.[7]

In order to characterise requests, we consider two specific relations over requests: $\mathsf{wf} \subseteq Q$ and $\rightsquigarrow \subseteq Q \times Q$. Intuitively, $\mathsf{wf}(q)$ indicates that q is well-formed, i.e., this request could be evaluated as such and $q \rightsquigarrow q'$ that q can be *extended* to q'. In other words, if $q \rightsquigarrow q'$, then it is possible to add information to q to obtain q'. For the sake of simplicity, we assume that $\rightsquigarrow$ only extends to well-formed requests, i.e., if $q \rightsquigarrow q'$, then $\mathsf{wf}(q')$ holds, regardless of whether $\mathsf{wf}(q)$ holds or not.

The relation wf can potentially include all possible requests, thus indicating that any request is evaluable as such. However, we can also use it to characterise the attribute domain. For instance, if a given attribute α_1 must always be present and could take any number of values, we could define wf as, for any request q:

$$\mathsf{wf}(q) \Leftrightarrow \exists v \ (\alpha_1, v) \in q$$

[7] The UN High Commissioner for Refugees estimate that at least 10 millions people are stateless: http://www.unhcr.org/pages/49c3646c155.html.

In this case, a request q not containing (α, v) would not be well-formed. Similarly, we could indicate that an attribute α_2 can have at most one value:

$$\mathsf{wf}(q) \Leftrightarrow \forall v, v' \ (\alpha_2, v) \in q \wedge (\alpha_2, v') \in q \Rightarrow v = v'$$

In some cases, the relation $\rightsquigarrow$ can be defined through an *administrative policy*, i.e., a policy describing how the security attributes can be changed within the system. The problem of knowing whether, given a security configuration, there exists a sequence of modifications allowing a given request to be granted is known as the safety problem [11]. In this paper, we focus on the fact that attributes might be missing, and we leave for future work the integration of administrative policies. Hence, a simple way to define $\rightsquigarrow$ is to use set inclusion, i.e., $q \rightsquigarrow q'$ if and only if $q \subseteq q'$ and $\mathsf{wf}(q')$, and unless specified otherwise, we consider this definition in the remainder of the paper.

Based on these two relations, a request q can be:

- not well-formed, in which case it should not be evaluated;
- well-formed and not extendable, i.e., there is no request q' such that $q \rightsquigarrow q'$, in which case we know that all information is present, and the request can be safely evaluated;
- well-formed and extendable, i.e., there exists a request q' such that $q \rightsquigarrow q'$, in which case the request is potentially both evaluable and missing information.

The last case corresponds to what we call *non-deterministic attribute retrieval*. Indeed, if a request is both well-formed and extendable, then it is not known whether the request should be evaluated as such (i.e., attribute retrieval is complete) or should be extended (i.e., attribute retrieval is not complete).

3.2 Requirements

We are now in position to establish some intuitive requirements for a general evaluation function $[\![\cdot]\!]$, such that given a policy p and a request q, $[\![p]\!](q)$ returns a set of decisions. The first requirement expresses the fact that any decision returned for a request where information is missing corresponds to a decision that would be returned had the missing information been provided.

Requirement 1 *Given a request q, if $d \in [\![p]\!](q)$, then there exists a request q' such that $q \rightsquigarrow q'$ and $[\![p]\!](q') = \{d\}$.*

The second requirement is the converse of the first one, and expresses that if a decision can be returned when all information in a request is provided, then evaluating the same request but with missing information should at least return that decision.

Requirement 2 *Given a request q, for any d such that there exists a request q' such that $q \rightsquigarrow q'$ and $[\![p]\!](q') = \{d\}$, $d \in [\![p]\!](q)$.*

Intuitively, these two requirements could correspond to the notions of correctness and completeness, respectively. Indeed, Requirement 1 implies that no incorrect decision can be returned when information is missing, i.e., no decision that could not have been returned had all information been provided. A trivial evaluation mechanism satisfying this requirement is that always returning the empty set of decisions for any request (i.e., a mechanism that never returns an incorrect decision). Conversely, Requirement 2 implies that *all* the correct decisions are returned when information is missing (and potentially more). A trivial mechanism satisfying this requirement is that returning all possible decisions for any query.

They are also somehow close to the notions of safety introduced in [22] or that of monotonicity introduced in [6], when the relation $\rightsquigarrow$ represents some ordering over the requests. However, these requirements are stronger because they impose the existence of a "higher" request returning a decision returned by the "lower" request.

Let us now consider again Alice, the security guard introduced at the start of the paper. If she receives the decisions $\{d_1, \ldots, d_n\}$ from the PDP, and if she knows that the PDP is meeting these two requirements, then she can deduce that all d_i are potentially correct decisions, had all information been provided, and that any decision different from any d_i is not a potentially correct decision. She can therefore reduce her choice to selecting a decision in $\{d_1, \ldots, d_n\}$. In particular, when the set returned by the PDP is reduced to a unique decision, then she can be certain that this decision is the correct one.

3.3 PTaCL Analysis

We now show that PTaCL satisfies neither Requirement 1 nor Requirement 2.

Firstly, consider the policy $p_3 = ((\mathbf{nat}, \mathsf{AT}), 1) \vartriangle ((\mathbf{nat}, \mathsf{AT}), 0)$, where $\vartriangle$ stands for the deny-overrides operator. Recall that $\vartriangle$ defines an ordering of decisions where 0 has precedence over 1 and, in turn, 1 has precedence over $\bot$. If we evaluate the empty request against p_3, we have

$$\begin{aligned} [\![((\mathbf{nat}, \mathsf{AT}), 1)]\!]_{\mathrm{P}}(\emptyset) &= \{1, \bot\} \\ [\![((\mathbf{nat}, \mathsf{AT}), 0)]\!]_{\mathrm{P}}(\emptyset) &= \{0, \bot\} \\ [\![p_3]\!]_{\mathrm{P}}(\emptyset) &= \{1, 0, \bot\}\,. \end{aligned}$$

However, it is easy to see that there is no request q such that $[\![p_3]\!]_{\mathrm{P}}(q) = \{1\}$, since if $((\mathbf{nat}, \mathsf{AT}), 1)$ evaluates to 1, then $((\mathbf{nat}, \mathsf{AT}), 0)$ necessarily evaluates to 0. In other words, regardless of the exact definition of $\rightsquigarrow$, even though $1 \in [\![p_3]\!]_{\mathrm{P}}(q)$, there is no request q' such that $q \rightsquigarrow q'$ and $[\![p_3]\!](q') = \{1\}$. Thus, PTaCL does not satisfy Requirement 1.

Let us now consider the policy $p_1 = \neg \sim \neg((\mathbf{nat}, \mathsf{AT}), 0)$, defined in Section 2.2, the request $q = \{(\mathbf{nat}, \mathsf{FR})\}$ and $q' = \{(\mathbf{nat}, \mathsf{FR}), (\mathbf{nat}, \mathsf{AT})\}$, and let us consider that $q \rightsquigarrow q'$. In other words, we consider that it is possible to add another nationality attribute to a request already containing one.

As described before, we have $[\![p_1]\!](q) = \{0\}$, while $[\![p_1]\!](q') = \{1\}$. In other words, the decision returned by the extended request q' does not appear in the set returned by the request q; thus, we can conclude that PTaCL does not satisfy Requirement 2. Note that this kind of situation is described in [6] as *partial attribute hiding*, and relies on the inability to know, when multiple values are allowed for a single attribute, whether all the values have been provided or not. Note that this inability extends to "real-world" situations: if someone presents a valid passport at a border control, there is no general way for the border officer to know whether this person has an additional citizenship.

In other words, the decision set received by an access control resolver is not necessarily helpful to make a conclusive decision. It is worth observing that these two observations also hold for XACML 3.0. One possibility is to constrain the policy language: if the policy is constructed using some specific constraints, then some monotonicity results can be shown [6]. In this paper, we propose a new approach by designing an evaluation function explicitly relying on the non-determinism of the attribute requests.

3.4 Evaluating with non-deterministic attribute retrieval

We now introduce a new method for computing the set of decisions returned by the evaluation of a PTaCL policy (t, p) for a request q. Informally, we remove any indeterminism from the evaluation of targets, instead evaluating a set of associated requests (specifically those that are extensions of q). More formally, we define

$$[\![(n, v)]\!]_{\mathrm{C}}(q) = \begin{cases} 1 & \text{if } (n, v') \in q \text{ and } v = v', \\ 0 & \text{otherwise.} \end{cases}$$

$$[\![(t, p)]\!]_{\mathrm{C}}(q) = \begin{cases} [\![p]\!]_{\mathrm{C}}(q) & \text{if } [\![t]\!]_{\mathrm{C}}(q) = 1, \\ \bot & \text{otherwise,} \end{cases}$$

Thus, the evaluation of a target is either 0 or 1, where 1 is returned if there exists a matching attribute and 0 is returned otherwise. The evaluation of a request with respect to a policy p guarded by a target t is $\bot$ if the evaluation of the target is 0 (that is, the policy is not applicable to the request), otherwise the result of evaluating p is returned. Composite targets and policies are evaluated by applying the corresponding operators to the results of evaluating their respective operands.

Given a request q, we define the set of extensions to q, denoted by $Ext(q)$ to be $\{q' \in Q \mid q \rightsquigarrow q'\}$. For the sake of simplicity, we assume here that for a given request q, $Ext(q)$ is finite, and we leave for future work the study of infinite sets of request extensions. Then

$$[\![p]\!]_{\mathrm{N}}(q) = \bigcup_{q' \in Ext(q)} [\![p]\!]_{\mathrm{C}}(q')$$

Thus we have non-determinism in request evaluation, as with the original evaluation method used in PTaCL. However, the non-determinism now arises because we consider all possible related requests and the decisions associated with those requests. It is clear that this new evaluation method satisfies Requirements 1 and 2.

We say that this approach use *non-deterministic attribute retrieval*, since, in general, when evaluating a request that does not have a given (α, v), we do not know whether this value should have been retrieved or not, and we should therefore perform the evaluation taking into account both possibilities.

4 Evaluating with PRISM

We show in this section how to define the evaluation functions $[\![\cdot]\!]_{\mathrm{C}}$ and $[\![\cdot]\!]_{\mathrm{N}}$ using the probabilistic model-checker PRISM [13]. We only present here the basic PRISM elements used for our encoding, and we refer the reader to the manual[8] for further details.

4.1 PRISM

Intuitively, a PRISM model consists of several modules, which can be synchronised. Each module can contain some variables, and describes the possible transitions at each step. A transition has the following general form:

```
[] g → p1:(post1) + ... + pn:(postn)
```

where g is a Boolean expression representing the guard of the transition, and p_i is the probability that the post-condition (i.e., some conditions on the variables) post_i is selected. For instance, a very simple example of a module representing a coin toss can be defined as:

```
module coin_toss
  head : bool init false;
  [] true → 0.5:(head'=true) +  0.5:(head'=false);
endmodule
```

where (head'=**true**) is a post-condition, indicating that the value of the variable head after the transition (which is indicated by the apostrophe) is **true**. In this particular example, the coin is fair and the probability of setting head to **true** is the same than that of setting head to **false**, which effectively corresponds to obtaining tail. In general, when several transitions are enabled within a given module, one is non-deterministically selected.

The *state* of a model consists of a tuple containing all the variables defined in all the modules. A transition from one state to another can be fired if there is a matching transition in one of the modules for which the guard is true. A *path* is a (potentially infinite) succession of states, starting from the initial state, and firing one transition at the time. As we describe later, the aim of model-checking is usually to automatically check whether the paths of a model satisfy some properties.

[8] Available at `http://www.prismmodelchecker.org/`

4.2 Policy Translation

The basic principle behind our encoding of the PTaCL language in PRISM is that for each attribute α and each value v, the state contains a Boolean variable α_v indicating whether α is assigned the value v or not. Targets are defined as Boolean formulae over the state, and policies are defined as formulae in a three-valued logic.

There are two non trivial steps in our encoding: we need to implement the relation $\rightsquigarrow$ in order to describe how information can be added in the query; and we need to encode the three-valued logic into PRISM.

Attributes We now present the definition of the relation $\rightsquigarrow$ in PRISM. Here, we consider the case where $\rightsquigarrow$ is the set inclusion relation, i.e., any attribute value can be potentially added to a request, and that the relation wf contains all possible requests.

For each atomic target (α, v), we generate the following module.

```
module att_α_v
    α_v: bool init false;
    first_α_v: bool init false;

    [] !first_α_v → (α_v'=true) & (first_α_v'=true);
    [] !first_α_v → (α_v'=false) & (first_α_v'=true);
endmodule
```

Intuitively, the attribute α has the value v in the current state if and only if, after executing the internal transition, α_v is true. The Boolean variable first_α_v indicates that the choice to either assign v to α has been made. Note that this choice is indeed non-deterministic, since the module can choose either transition. This encoding corresponds to the relation $\rightsquigarrow$ since a request q can be extended to q' by adding one after the other the attribute values in $q' \setminus q$.

In addition, the formula all_first is defined such that it is true when all first_α_v are true, for any α and any v. In this case, the request corresponding to a state where all_first holds is no longer extendable.

Interestingly, the structure of PTaCL allows us to only care about atomic targets that are present in a policy, rather than all the possible ones [10]. This property holds because, at the moment, PTaCL only handles simple attribute matching and not complex matching. In other words, we can only check whether a value is present or not, and not whether the value of an attribute is less or equal than a value, for instance.

Following the definition of the function $[\![\cdot]\!]_C$, a target evaluates to a Boolean value: an atomic target (α, v) is true if and only if the variable α_v is true, and composite targets are evaluated by applying their operators on the evaluation of the sub-targets.

Policies evaluate to a value in $\{0, 1, \perp\}$, and therefore we cannot encode a policy directly as a Boolean formula. Instead, we encode the set $\{0, 1, \perp\}$ as the integer values 0, 1 and 2, respectively, and we adapt the logical operators

accordingly. We also use the PRISM ternary operator `c ? e1 : e2` which evaluates to `e1` if `c` is true and to `e2` otherwise. Given a PTaCL policy p, we note $\widehat{p}$ the PRISM expression corresponding the encoding of the policy p, which is defined as follows:

- $\widehat{0} = 0$ and $\widehat{1} = 1$;
- $\widehat{\neg p}$ is translated to $(\widehat{p} = 2)$? 2 : $(1 - \widehat{p}))$".
- $\widehat{\sim p}$ is translated to $\mathsf{mod}(\widehat{p}, 2)$.
- $\widehat{p_1 \sqcap p_2}$ is translated to $(\widehat{p_1} = 2 \mid \widehat{p_2} = 2)$? 2 : $(\widehat{p_1} * \widehat{p_2})$.
- $\widehat{p_1 \sqcup p_2}$ is translated to $\mathsf{max}(\widehat{p_1}, \widehat{p_2})$.
- $\widehat{p_1 \tilde{\sqcap} p_2}$ is translated to $\mathsf{min}(\widehat{p_1} * \widehat{p_2}, 2)$.
- $\widehat{p_1 \tilde{\sqcup} p_2}$ is translated to $(\widehat{p_1} = 1 \mid \widehat{p_2} = 1)$? 1 : $\mathsf{max}(\widehat{p_1}, \widehat{p_2})$.
- $\widehat{(t, p)}$ is translated to $\widehat{t}$? $\widehat{p}$: 2

This encoding of the policy ensures that if the state of the model corresponds of a request q, then $[\![p]\!]_{\mathrm{C}}(q) = d$ if and only if the formula p is equal to d. Note that a given policy can only evaluate to a single decision, as it is the case with $[\![\cdot]\!]_{\mathrm{C}}$. For instance, we give below the PTaCL definition for the policy p_1 defined in Section 2.2[9]:

```
t1 :: (Tatom "nat" "AT")
pzero : Patom Zero
p1_0 : Ptar t1 pzero
p1_1 : Pnot p1_0
p1_2 : Pdbd p1_1
p1 : Pnot p1_2
```

The intermediary policies are introduced for the sake of readability. This policy is automatically translated to the following PRISM model:

```
module att_nat_AT
        nat_AT: bool init false;
        first_nat_AT: bool init false;
        [] !first_nat_AT → (nat_AT'=true) & (first_nat_AT'=true);
        [] !first_nat_AT → (nat_AT'=false) & (first_nat_AT'=true);
endmodule

formula all_first = true & first_nat_AT;
formula t1 = (nat_AT);
formula pzero = 0;
formula p1_0 = ((t1) ? pzero: 2);
formula p1_1 = (p1_0 = 2 ? 2 : (1 − p1_0));
formula p1_2 = mod(p1_1, 2);
formula p1 = (p1_2 = 2 ? 2 : (1 − p1_2));
```

[9] We use here the syntax used for the tool ATRAP [10], which is quite straight-forward, and where `Ptar` is the constructor for target policies, `Pnot` for the operator $\neg$, and `Pdbd` is the operator for $\sim$.

In the initial state, which corresponds to the empty request, nat_AT is **false**, and we can see that p1 evaluates to 0. If the model takes the first transition of the module att_nat_AT, then nat_AT becomes **true**, which corresponds to the request $\{(\mathbf{nat}, \mathsf{AT})\}$, and we can see that the formula p1 now evaluates to 1.

4.3 Policy Evaluation

Given a PRISM model, we can specify properties in a temporal logic and automatically verify them. Roughly speaking, we can specify properties over the paths of the model, i.e., the sequences of possible states from the initial states. In the encoding proposed so far, a state contains two Boolean variables for each pair (α, v), the first indicating if α is associated with the value v, and the second if the choice has been made.

Given a path $p = (s_0, \ldots, s_n)$, where each s_i represents a state, and a property φ over states, there are two types of path properties: F φ and G φ. The former holds if there exists i such that for any $j \geqslant i$, $\varphi(s_j)$ holds, while the latter holds if $\varphi(s_i)$ holds for any i. Note that some paths can be infinite in general, although in our encoding, all paths are finite, since once the choice has been made for all attributes, there is no transition possible.

PRISM also supports the CTL operators A and E, such that given a path property ϕ, E ϕ holds if and only if there exists a path in the system for which ϕ holds, and A ϕ holds if and only if ϕ holds for all paths in the system.

As explained above, the evaluation of the formula p in a state corresponding to a request q effectively corresponds to $[\![p]\!]_{\mathrm{C}}(q)$. In addition, the relation $\rightsquigarrow$ is encoded through the transition of the attribute modules (since we consider here that $\rightsquigarrow$ is the set inclusion relation). In order to evaluate $[\![p]\!]_{\mathrm{N}}(q)$, we therefore want to start from the state corresponding to q, perform all possible extensions from this state, and each time, check which decision can be obtained.

In order to check which decisions can be reached from the state corresponding to a particular request, we use the PRISM operator filter, such that filter(op, prop, states) computes the value of the property prop for each state satisfying states, and combines these values using the operator op. More precisely, given a policy p and a request q, for each decision d, we evaluate the formula filter(exists, E [F all_first & $\widehat{p} = d$], $\widehat{q}$), where $\widehat{p}$ is the translation of the policy described above, and $\widehat{q}$ corresponds to the conjunction of all α_v such that $(\alpha, v) \in q$ and of all $!\alpha'\_v'$ such $(\alpha', v') \notin q$. This formula can be read as "does there exists a state where $\widehat{q}$ holds such that, there exists a path from that state where, eventually, after performing non-deterministic attribute assignment, $\widehat{p} = d$".

For instance, the evaluation of the policy p_1 is given in Table 3. The first row checks which decisions are reachable from a request containing $(\mathbf{nat}, \mathsf{AT})$, and as expected, only the decision 0 is reachable. The second row checks which decisions are reachable from an extendable request not containing $(\mathbf{nat}, \mathsf{AT})$, and both decision 0 and 1 are reachable, which corresponds to the evaluation of the empty request in Table 2. Finally, the last row checks which decisions are reachable from a non-extendable request not containing $(\mathbf{nat}, \mathsf{AT})$, and in this case, only the decision 1 is reachable, since there is no possibility to add

Table 3: Evaluation of p_1 in PRISM

Property	$d = 0$	$d = 1$	$d = 2$
filter(exists, E [F all_first & p1= d], nat_AT)	**true**	**false**	**false**
filter(exists, E [F all_first & p1= d], !nat_AT)	**true**	**true**	**false**
filter(exists, E [F all_first & p1= d], !nat_AT & first_nat_AT)	**false**	**true**	**false**

(**nat**, AT) in such a request. Hence, a strength of this evaluation mechanism is that it allows us to distinguish between a request to which it is not sure if a given attribute value belongs and a request to which we are certain a given attribute value does not belong.

It is worth observing that the situation described in Section 3.3 to show that PTaCL does not satisfy Requirement 2 does not apply with the PRISM evaluation. Indeed, the problem came from the fact that the request $\{(\mathbf{nat}, \mathsf{FR})\}$ evaluates to $\{0\}$, while $\{(\mathbf{nat}, \mathsf{FR}), (\mathbf{nat}, \mathsf{AT})\}$ evaluates to $\{1\}$. However, with the PRISM evaluation, the former request is equivalent to the empty request, since it does not contain (**nat**, AT), and therefore evaluates both to 0 and 1.

Similarly, if we evaluate the policy p_3 defined in Section 3.3 to show that PTaCL does not satisfy Requirement 1, we have that both decisions 0 and 2 are reachable from the empty request, but not decision 1.

5 Probabilistic Attribute Retrieval

5.1 Specific probabilities

In the previous section, attribute retrieval was non-deterministic, since PRISM could choose to assign a value to an attribute or not. It is also possible to associate specific probabilities with a particular value for a particular attribute. For instance, we could say that for a given population of users, the likelihood of a user to have the Austrian nationality is 0.6.

Hence, we introduce in PTaCL the possibility to define the probability of an attribute to receive a given value through the construction attribute a v p, which declares that the probability of the attribute a to be associated with the value v equals p. For instance, we can add to the previous example the statement attribute "att" "AT" 0.6. The generated code is then:

```
const double p_nat_AT = 0.6;

module att_nat_AT
 nat_AT: bool init false;
 first_nat_AT: bool init false;
 [] !first_nat_AT → p_nat_AT:(nat_AT'=true) & (first_nat_AT'=true)
                  + (1 − p_nat_AT):(nat_AT'=false) & (first_nat_AT'=true);
endmodule
```

Table 4: Evaluation of p_1 in PRISM with probability of $(\mathbf{nat}, \mathsf{AT}) = 0.6$.

Property	$d = 0$	$d = 1$	$d = 2$
filter(range, P =? [F all_first & p1= d], nat_AT)	[1.0,1.0]	[0.0,0.0]	[0.0,0.0]
filter(range, P =? [F all_first & p1= d], !nat_AT)	[0.0,0.6]	[0.4,1.0]	[0.0,0.0]
filter(range, P =? [F all_first & p1= d], !nat_AT & first_nat_AT)	[0.0,0.0]	[1.0,1.0]	[0.0,0.0]

5.2 Probabilistic properties

PRISM is a probabilistic model-checker, which means that in addition to checking the temporal properties described above, PRISM can also check probabilistic properties, or rather, check properties over a probabilistic system. In particular, PRISM provides the operator P =?, such that, given a path property ϕ, P =? ϕ returns the probability of this property to hold, i.e., returns the aggregated probabilities of all paths for which ϕ holds.

In this case, instead of checking whether a particular decision d can be reached from a particular state q, we can check the probability of that decision to be reached with the property filter(range, P =? [F all_first & $\widehat{p}$ =d], $\widehat{q}$). Note that we use the operator range instead of exists, since the value returned by the operator P =? is a probability. Hence, this formula returns the minimal and maximal probabilities to reach d from states satisfying $\widehat{q}$.

For instance, Table 4 provides the PRISM evaluation for the policy p_1 when the attribute **nat** can be assigned to AT with probability 0.6. Perhaps the most interesting row is the second one, which indicates that the probability of reaching the decision 0 from a state where nat_AT does not hold ranges in [0.0, 0.6]. This result is explained by the fact that there are two states where nat_AT does not hold: the state where the choice not to assign AT to **nat** has already been made, from which the probability to reach 0 is null, and the state where this choice has not been made, from which the probability of reaching 0 is 0.6. A similar reasoning can be performed for the probability range of reaching the decision 1.

In order to illustrate the interest of this approach, let us again consider Alice, who has to decide to let Bob enter the building or not. If Bob has not provided any information about his nationality, then if Alice uses the evaluation mechanism presented in Section 4.3, she only knows that both 0 and 1 are possible decisions. On the other hand, if she knows that the probability of Bob to have the Austrian nationality is 0.6, she can analyse the probability ranges for each decision, and therefore make a more informed decision.

5.3 Mixing non-deterministic and probabilistic attribute retrieval

The retrieval of a value to an attribute can either be seen as non-deterministic (as in Section 4.2), in which case we can check if a path property is true for at least one path. It can also be probabilistic (as in Section 5), in which case we can check the aggregated probability that a path property holds.

Interestingly, we can combine both types of attributes, i.e., have a system where some attributes are non-deterministically assigned, and other are probabilistically assigned. However, in this case, the operator P =? can no longer be used, and the operators Pmin =? and Pmax =? should be instead, indicating the minimum and maximum probability of reaching a decision, based on the non-deterministic choices. This distinction is somehow close to that used when using the operator range in the previous section. The study of mixed non-deterministic and probabilistic attribute retrieval is an ongoing work.

6 Conclusion

In this paper, we have investigated the problem of attribute retrieval in Attribute based Access Control (ABAC), and in particular the fact that the evaluation function of PTaCL and XACML can return multiple decisions when some attribute values are missing. We have shown that these multiple decisions are not necessarily consistent with the decisions that would have been returned had all attribute values been provided. We have designed a new evaluation mechanism that is consistent with this intuition, and demonstrated how to encode PTaCL policies into the probabilistic model-checker PRISM in order to compute this evaluation mechanism.

We have also shown how probabilistic attribute retrieval can be defined. We believe this can be particularly useful in order to resolve decision sets returned by the PDP. A very interesting aspect of PRISM is its ability to mix non-deterministic and probabilistic attribute retrieval, which allows for a "best effort" strategy: in general, we might not know the probability of all attribute values, but by specifying those that we know, we can get a somehow more accurate analysis.

We currently support an automatic translation of PTaCL policies into PRISM, together with the generation of the path properties required for evaluation. We are still in the process of conducting performance analyses, but it is worth noting that PRISM should be able to handle model with 10^7 to 10^8 states on a typical PC[10], so scalability is not necessarily a pressing issue.

There are multiple leads for future work, including the integration of complex policy analyses, such as policy safety [10], including a probabilistic aspect, or the integration of administrative policies. Another interesting lead is to investigate the influence of the presentation of complex decisions to the access control resolver, for instance using nudging, or other psychological influencing techniques [15]. In particular, we know that human users are not particularly good at understanding probabilities when making choices [23].

Acknowledgement This work has been partially funded by the Dutch national program COMMIT under the THeCS project.

[10] http://www.prismmodelchecker.org/manual/FrequentlyAskedQuestions/PRISMModelling

References

1. D. E. Bell and L. J. LaPadula. Secure computer systems: A mathematical model, Volume II. *Journal of Computer Security*, 4(2/3):229–263, 1996.
2. P. Bonatti, S. De Capitani Di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security*, 5(1):1–35, 2002.
3. D. F. C. Brewer and M. J. Nash. The Chinese Wall Security Policy. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 329–339, May 1989.
4. G. Bruns and M. Huth. Access control via Belnap logic: Intuitive, expressive, and analyzable policy composition. *ACM Transactions on Information and System Security*, 14(1):9, 2011.
5. J. Crampton and M. Huth. An authorization framework resilient to policy evaluation failures. In D. Gritzalis, B. Preneel, and M. Theoharidou, editors, *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 472–487. Springer, 2010.
6. J. Crampton and C. Morisset. PTaCL: A language for attribute-based access control in open systems. In *POST 2012, Proceedings*, volume 7215 of *Lecture Notes in Computer Science*, pages 390–409, 2012.
7. J. Crampton and C. Morisset. Monotonicity and completeness in attribute-based access control. In S. Mauw and C. D. Jensen, editors, *Security and Trust Management - 10th International Workshop, STM 2014, Wroclaw, Poland, September 10-11, 2014. Proceedings*, volume 8743 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2014.
8. D. F. Ferraiolo and D. R. Kuhn. Role-based access control. In *Proceedings of the 15th National Computer Security Conference*, pages 554–563, 1992.
9. K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 196–205, New York, NY, USA, 2005. ACM.
10. A. Griesmayer and C. Morisset. Automated certification of authorisation policy resistance. In J. Crampton, S. Jajodia, and K. Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 574–591. Springer, 2013.
11. M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.
12. S. Kleene. *Introduction to Metamathematics*. D. Van Nostrand, Princeton, NJ, 1950.
13. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
14. B. Lampson. Protection. In *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, pages 437–443, Princeton University, 1971.
15. C. Morisset, T. Groß, A. P. A. van Moorsel, and I. Yevseyeva. Nudging for quantitative access control systems. In T. Tryfonas and I. G. Askoxylakis, editors, *Human Aspects of Information Security, Privacy, and Trust - Second International Conference, HAS 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings*, volume 8533 of *Lecture Notes in Computer Science*, pages 340–351. Springer, 2014.
16. C. Morisset and N. Zannone. Reduction of access control decisions. In *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies*, SACMAT '14, pages 53–62, New York, NY, USA, 2014. ACM.

17. OASIS. *eXtensible Access Control Markup Language (XACML) Version 2.0*, 2005. Committee Specification.
18. OASIS. *eXtensible Access Control Markup Language (XACML) Version 3.0*, 2010. Committee Specification 01.
19. S. Ranise and A. T. Truong. Incremental analysis of evolving administrative role based access control policies. In V. Atluri and G. Pernul, editors, *Data and Applications Security and Privacy XXVIII - 28th Annual IFIP WG 11.3 Working Conference, DBSec 2014, Vienna, Austria, July 14-16, 2014. Proceedings*, volume 8566 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2014.
20. S. Ranise, A. T. Truong, and A. Armando. Scalable and precise automated analysis of administrative temporal role-based access control. In S. L. Osborn, M. V. Tripunitara, and I. Molloy, editors, *19th ACM Symposium on Access Control Models and Technologies, SACMAT '14, London, ON, Canada - June 25 - 27, 2014*, pages 103–114. ACM, 2014.
21. P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo. An algebra for fine-grained integration of XACML policies. In B. Carminati and J. Joshi, editors, *SACMAT*, pages 63–72. ACM, 2009.
22. M. C. Tschantz and S. Krishnamurthi. Towards reasonability properties for access-control policy languages. In D. F. Ferraiolo and I. Ray, editors, *SACMAT*, pages 160–169. ACM, 2006.
23. A. Tversky and D. Kahneman. The framing of decisions and the psychology of choice. *Science*, 211(4481):453–458, 1981.
24. D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. *ACM Transactions on Information and System Security*, 6(2):286–235, 2003.
25. N. Zhang, M. Ryan, and D. P. Guelev. Evaluating access control policies through model checking. In J. Zhou, J. Lopez, R. H. Deng, and F. Bao, editors, *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings*, volume 3650 of *Lecture Notes in Computer Science*, pages 446–460. Springer, 2005.