Analysis of XACML policies with SMT

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science

Analysis of XACML Policies with SMT

Fatih Turkmen, Jerry den Hartog, Silvio Ranise and Nicola Zannone

14/08

ISSN 0926-4515

editors: prof.dr. P.M.E. De Bra
prof.dr.ir. J.J. van Wijk

Reports are available at:
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Author&level=1 and
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Year&Level=1

Computer Science Reports 14-08
Eindhoven, November 2014

Analysis of XACML Policies with SMT

Fatih Turkmen[1], Jerry den Hartog[1], Silvio Ranise[2], and Nicola Zannone[1]

[1] Eindhoven University of Technology, Eindhoven, Netherlands
[2] Fondazione Bruno Kessler (FBK) Trento, Italy

Abstract. The eXtensible Access Control Markup Language (XACML) is an extensible and flexible XML language for the specification of access control policies. However, the richness and flexibility of the language (along with the verbose syntax of XML) come with a price: errors are easy to make and difficult to detect when policies grow in size. If these errors are not detected and rectified, they can result in serious data leakage and/or privacy violations leading to significant legal and financial consequences. To assist policy authors in the analysis of their policies, several policy analysis tools have been proposed based on different underlying formalisms. However, most of these tools either abstract away functions over non-Boolean domains (hence they cannot provide information about them) or produce very large encodings which hinder the performance. In this paper, we present a generic policy analysis framework that employs SMT as the underlying reasoning mechanism. The use of SMT does not only allow more fine-grained analysis of policies but also improves the performance. We demonstrate that a wide range of security properties proposed in the literature can be easily modeled within the framework. A prototype implementation and its evaluation are also provided.

1 Introduction

Access rules governing sensitive data such as patient health records or financial transactions are usually encoded in a policy that is enforced by the authorization system. Correctness of the access control policies is crucial for organizations to prevent authorization violations or fraud which can result in serious data leakage and/or privacy violations leading to significant legal and financial consequences (e.g., financial and reputation loss). In this work, we consider policies expressed in eXtensible Access Control Markup Language (XACML) [24]. XACML provides an extensible and flexible language that allows the specification of structured policies in which the policies specified by different authorities can be combined together. However, policy specification in XACML is known to be a difficult and error-prone task [11, 16]. This richness and flexibility along with its verbose syntax make it difficult to determine whether policies work as intended. Therefore, automated tools are needed to assist policy authors in analyzing their policies to detect and correct errors before policies are deployed.

This need has spurred the development of several methods and tools for the verification of policy specifications at design time using formal reasoning [4, 8, 11, 15, 16]. The security properties being verified can express requirements on the

policies but also on relations between policies. A requirement on a policy could be (types of) access requests that should (not) be granted by the policy. An updated policy being compared with the original to ensure the update is 'safe' is an example of a requirement on the relation. Here 'safe' could be expressed in being as permissive/restrictive as another policy as is done in the properties policy refinement [4] and subsumption [16]. Despite a large variety in security properties that one may need to check, existing policy analysis tools often support only a restricted set of properties due to the (lack of) expressiveness of the formalization employed by the tool and the capabilities offered by the underlying reasoner.

Advances in propositional satisfiability (SAT) research [13] make SAT solvers an attractive underlying reasoner in policy analysis [16]. SAT allows efficient reasoning about propositional logic formula and many access control policies and security properties can be naturally modeled in propositional logic. However, SAT solvers do not natively support reasoning on predicates over non-Boolean variables and functions which frequently appear in access control policies and, in particular, in XACML policies. For instance, SAT does not allow a straightforward reasoning on temporal constraints such as *request-time* $>$ 13:20, which can play an important role in the correctness of a policy and thus in the security of the system. Such non-Boolean expressions are usually left uninterpreted [16] which restricts analysis capabilities. Alternatives that support fine-grained policy analysis can lead to excessively large encodings of the policy. The analyst is forced to choose a trade-off between performance and accuracy by introducing bounds on the domains.

In this paper, we consider SAT modulo theories (SMT) [5] as the underlying reasoning method for the analysis of XACML policies. SMT enables the use of theories, such as linear arithmetic and equality, to reason about the satisfiability of first order formulas. SMT is a natural extension to SAT in which SMT solvers employ tailored reasoners when solving non-Boolean predicates in the input formula. The use of SMT makes it possible to perform a more fine-grained analysis than existing SAT-based policy analysis tools allow.

The contributions of this paper are thus as follows:

- A novel policy analysis framework which makes it possible to verify access control policies against a large range of security properties.
- A fine-grained analysis of access control policies by performing reasoning on non-Boolean predicates, e.g. arithmetic functions on numeric attributes.
- A prototype implementation of the framework and its extensive evaluation using a number of well-known security properties taken from the literature.

The remainder of the paper provides an overview of XACML and SMT in Section 2, and an encoding of XACML policies in SMT in Section 3. Our analysis framework that uses this encoding for policy analysis is given in Section 4. Section 5 presents a prototype of our framework with experimental results. Section 6 discusses related work and Section 7 provides conclusions.

2 Preliminaries

In this section we shortly recall key points of XACML and SMT.

2.1 XACML

XACML [24] is an OASIS standard for the specification of access control policies. It provides an attribute-based language that allows the specification of composite policies. Three policy elements are provided by XACML: *policy sets*, *policies* and *rules*.

A policy set consists of policy sets and policies; policies in turn consist of rules. If policy element p_1 is nested in policy element p_2 we say that p_1 is a *child policy element* of p_2 and that p_2 is the *parent policy element* of p_1. Each policy element has a (possibly empty) *target* which defines (restricts) the applicability of the policy element in terms of attributes characterizing the subject, the resource, the action to be performed on the resource, and the environment. Intuitively, the target identifies the set of access requests that the policy element applies to. In addition, rules specify an *effect* element that defines whether the requested actions should be allowed (*Permit*) or denied (*Deny*), and can be associated with *conditions* to further restrict their applicability.

If an access request matches both the target and conditions of a rule, the rule is applicable to the request and yields the decision specified by its effect element. Otherwise, the rule is not applicable, and a *NotApplicable* decision is returned. If an error occurs during evaluation, an *Indeterminate* decision is returned. In some cases, an extended set of *Indeterminate* values is used to allow for a fine-grained combination of decisions: *Indeterminate*$\{P\}$, *Indeterminate*$\{D\}$ and *Indeterminate*$\{PD\}$. Intuitively, these *Indeterminate* decisions indicate the evaluation result of the policy element if the error not occurred.

To combine decisions obtained from the evaluation of different applicable policy elements, XACML provides a number of combining algorithms [24]: permit-overrides, deny-overrides, deny-unless-permit, permit-unless-deny, first-applicable and only-one-applicable.[3] Intuitively, these algorithms define procedures to evaluate composite policies based on the order of the policy elements and priorities between decisions.

Next we present a sample XACML policy in a concise form that we will use as a running example through the paper.

Example 1 *A user is allowed to create an object of type "transaction" only if his credit balance (*credit*) is higher than the value of the transaction itself (*value*) and banking costs (*cost*). Transactions can only be created during working days (i.e., Monday, Tuesday, Wednesday, Thursday, Friday) within the time interval*

[3] Combining algorithms permit-overrides and deny-overrides are defined over the *Indeterminate* extended set, while the other algorithms are defined over a single *Indeterminate* decision value. Combining algorithm only-one-applicable can be only used to combine policy sets and policies.

08:00-18:00. One way to model this policy is to represent (the negation of) these constraints as Deny *rules and then to combine the resulting rules using* deny-overrides *(*dov*):*

$$
\begin{array}{ll}
p[\mathsf{dov}]: & \mathsf{resource\text{-}type} = \text{“}transaction\text{”} \wedge \mathsf{action\text{-}id} = \text{“}create\text{”} \\
r_1[Deny]: & \mathsf{value} + \mathsf{cost} > \mathsf{credit} \\
r_2[Deny]: & \mathsf{current\text{-}day} \notin \{Mo, Tu, We, Th, Fr\} \vee \\
 & \mathsf{current\text{-}time} < 08{:}00 \vee \mathsf{current\text{-}time} > 18{:}00 \\
r_3[Permit]: & \textbf{true}
\end{array}
$$

where **true** *is used to indicate that the target of the policy element matches every access request. We assume that attributes* value*,* cost*,* credit*,* current-time *and* current-day *are further constrained with function* one-and-only *so that a policy element returns* Indeterminate *if multiple values are provided for them.*

2.2 Satisfiability Modulo Theories

SMT [5] is a generalization of SAT in which Boolean variables can be replaced by predicates from a variety of theories. To specify SMT formulas, we follow an extended version of the SMT-LIB (v2) standard (`http://www.smtlib.org`) which is based on many-sorted first order logic. In the following, we assume the usual syntactic (e.g., sort, constant, predicate and function symbols, terms, atoms, literals, Boolean connectives, quantifiers, and formulas) and semantic (e.g., structure, satisfaction, model, and validity) notions of many-sorted first order logic; see [9] for formal definitions.

A theory $\mathcal{T}$ consists of a signature and a class of models. Intuitively, the signature fixes the vocabulary to build formulas and the class of models gives the meaning of the symbols in the vocabulary. As an example, consider the theory of an enumerated data-type: the signature consists of a single sort symbol and n constants corresponding to the elements in the enumeration; the class of models contains all structures interpreting the sort symbol as a set of cardinality n. For Linear Arithmetic over the Integers (LAI), the signature consists of the numerals (corresponding to the integers), binary addition, and the usual ordering relations; the class of models contains the standard model of the integers in which only linear constraints are considered. For the theory of uninterpreted functions, the signature consists of a finite set of symbols and the equality sign; the class of models contains all those structures interpreting the equality sign as a congruence relation and the other symbols in the signature as arbitrary constants, functions, or relations.

A formula φ is *$\mathcal{T}$-satisfiable* (or *satisfiable modulo $\mathcal{T}$*) iff there exists a structure $\mathcal{M}$ in the class of models of $\mathcal{T}$ and a valuation ϕ (i.e. a mapping from the variables that are not in the scope of a quantifier in the formula to the elements in the domains of $\mathcal{M}$) satisfying φ (in symbols, $\mathcal{M}, \phi \models \varphi$). A formula φ is *$\mathcal{T}$-valid* (or *valid modulo $\mathcal{T}$*) iff for every structure $\mathcal{M}$ in the class of models of $\mathcal{T}$ and every valuation ϕ, we have that $\mathcal{M}, \phi \models \varphi$. Notice that a formula φ is $\mathcal{T}$-valid iff the negation of φ (i.e. $\neg\varphi$) is $\mathcal{T}$-unsatisfiable.

Checking the satisfiability of conjunctions of literals (i.e., atoms or their negations) modulo certain theories – e.g., the theory of uninterpreted functions, theories of enumerated data-types, and Linear Arithmetic over the Integers – is well-known to be decidable [5]. These results imply the decidability of checking the satisfiability of quantifier-free formulas modulo the same theories. This is so as it is always possible to transform arbitrary Boolean combinations of atoms into disjunctive normal form (DNF), i.e. in a disjunction of conjunctions of literals. Unfortunately, the transformation to DNF may be computationally expensive and generate an exponentially larger formula [9]. For this reason, even if checking the satisfiability of conjunctions of literals modulo certain theories is polynomial (as it is the case for the theory of uninterpreted functions), checking the satisfiability of quantifier-free formulas modulo the same theories becomes NP-hard. While these theoretical limitations are unavoidable, modern SMT solvers have developed a wealth of heuristics to scale and handle very large formulas with arbitrary Boolean structures. The interested reader is pointed to [5] for a thorough introduction.

The situation is further complicated by two possible sources of problems. First, several verification problems (such as the XACML policy analysis problems considered in this paper) require to consider more than one theory to model various aspects of the situation under scrutiny. Under suitable assumption on the component theories, it is possible to build theory solvers capable of checking the satisfiability of conjunctions of literals in combinations of theories by modularly re-using the theory solvers of the component theories .However, the complexity of checking the satisfiability of conjunctions of literals in the combination can be much higher than that of modulo the individual theories. For instance, there exists a combination of two theories with polynomial satisfiability problem whose combination becomes NP-complete [26]. The second source of problems is the presence of quantifiers in the proof obligations generated by certain verification tools (as it is the case of some of the policy analysis problems considered in this work). In fact, the decidability of quantifier-free formulas does not extend to quantified formulas. For instance, checking the satisfiability of quantified formulas modulo the theory of uninterpreted functions is undecidable since one can encode the satisfiability problem for arbitrary first-order formulas whose undecidability is well-known [9]. Despite this and other negative results, a lot of efforts have been put in identifying classes of quantified formulas whose satisfiability is decidable by integrating instantiation or quantifier-elimination procedures in SMT solvers; see, e.g., [5] for pointers to relevant works.

A more detailed discussion on the decidability and complexity of $\mathcal{T}$-satisfiability problems related to the analysis of XACML policies is provided in Appendix A.

3 Encoding XACML policies in SMT

In this section, we first present our formalization of XACML policies that allows us to represent policies in terms of predicates. We then show how the obtained predicates can be used to define SMT formulas.

3.1 XACML Formalization

An access control schema $\langle Att, Dom \rangle$ defines the vocabulary used for specifying access control policies. Here Att is a set of attributes $a_1, \ldots, a_n$, Dom gives the corresponding attribute domains $Dom_{a_1}, \ldots, Dom_{a_n}$ and we refer to set $2^{Dom_{a_1}} \times \ldots \times 2^{Dom_{a_n}}$ as the *policy space* specified within the schema. The elements of the policy space are called *attribute assignments*. An attribute assignment maps attributes to a (possibly empty) bag of values in their domains. An *access request* $\langle a_1 = v_1, \ldots, a_m = v_m \rangle$ specifies an attribute assignment by providing the values for those attributes not assigned the empty bag (multiple attribute/value pairs with the same attribute indicate multiple values are assigned to that attribute). Hereafter, $\mathcal{R}$ denotes the set of all possible access requests, i.e. the policy space.

Each policy element in XACML has a target that specifies *applicability constraints* in terms of attribute assignments. Applicability constraints are used to divide the policy space in three disjoint sub-spaces: the *Applicable* space AS_A, the *Indeterminate* space AS_{IN}, and the *NotApplicable* space AS_{NA}. These sub-spaces respectively represent access requests for which the policy's target matches the request, checking whether the target matches the request produces an error, and the target does not match the request. We represent the applicability space of a policy element as $\langle AS_A, AS_{IN} \rangle$ with an access request $req \in AS_{NA}$ iff $req \notin AS_A \cup AS_{IN}$. An access request is evaluated against a policy element only if it matches the target of policy element's parent. Based on this observation, we flatten a XACML policy by propagating its applicability constraints in a top-down fashion from the root policy element to rules.

Definition 1 *Let p be a policy with applicability space $\langle AS^p_A, AS^p_{IN} \rangle$ and q a child policy element of p. Let $\langle AS^T_A, AS^T_{IN} \rangle$ be the applicability space induced by the target of q. The applicability space of q is defined as follows:*
$AS^q_A = AS^T_A \cap AS^p_A$
$AS^q_{IN} = (AS^T_{IN} \cap AS^p_A) \cup AS^p_{IN}$

Example 2 *Consider the policy in Example 1. Below we represent the applicability constraints defined from the target of every policy element:*
ac_0 : "*transaction*" $\in$ resource-type
ac_1 : "*create*" $\in$ action-id
ac_2 : $\bigwedge_{d \in \{Mo,Tu,We,Th,Fr\}} d \notin$ current-day
ac_3 : $\forall v \in$ current-time $v > 18{:}00$
ac_4 : $\forall v \in$ current-time $v < 8{:}00$
ac_5 : $\forall v_1 \in$ credit, $v_2 \in$ cost, $v_3 \in$ value $(v_1 < v_2 + v_3)$
$ac_6, ..., ac_{10}$: $\forall v \notin att \vee (v_1 \neq v_2 \wedge v_1 \in att \wedge v_2 \in att)$
$(att \in \{$current-day, current-time, credit, cost, value$\})$
Constraints $ac_6, ..., ac_{10}$ address Indeterminate *cases by requiring att to be either empty (no value v belongs to att) or to contain at least two distinct elements (denoted by constants v_1 and v_2). The applicability space induced by the target of rule r_i, $\langle AS^{T_i}_A, AS^{T_i}_{IN} \rangle$, can be represented as follows (for the sake of simplicity, we represent sets of access requests as the applicability constraints that render*

them):
$T_1 : \langle ac_5, ac_8 \cup ac_9 \cup ac_{10} \rangle$
$T_2 : \langle ac_2 \cup ac_3 \cup ac_4, ac_6 \cup ac_7 \rangle$
$T_3 : \langle \mathcal{R}, \emptyset \rangle$
Policy p has applicability space $\langle ac_0 \cap ac_1, \emptyset \rangle$*; this space has to be propagated to rules. Thus, the applicability space* $\langle AS_A^{r_i}, AS_{IN}^{r_i} \rangle$ *of rule* r_i *is:*
$r_1 : \langle ac_0 \cap ac_1 \cap ac_5, (ac_8 \cup ac_9 \cup ac_{10}) \cap (ac_0 \cap ac_1) \rangle$
$r_2 : \langle ac_0 \cap ac_1 \cap (ac_2 \cup ac_3 \cup ac_4), (ac_6 \cup ac_7) \cap (ac_0 \cap ac_1) \rangle$
$r_3 : \langle ac_0 \cap ac_1, \emptyset \rangle$

Based on the possible decisions in XACML, the policy space can be partitioned into four disjoint subsets DS_P, DS_D, DS_{IN} and DS_{NA} by using rule effects and applicability constraints. These subsets represent the classes of access requests that evaluate to same access decision: *Permit*, *Deny*, *Indeterminate* and *NotApplicable*, respectively. We denote the decision space of a policy as $\langle DS_P, DS_D, DS_{IN} \rangle$. If an access request does not fall in $DS_P \cup DS_D \cup DS_{IN}$, then it falls in DS_{NA}. The decision space of a rule can be derived from its effect and applicability constraints.

Definition 2 *Let* $\langle AS_A, AS_{IN} \rangle$ *be the applicability space of a rule* r *and* Effect *its effect. The decision space of* r*, denoted* $\langle DS_P, DS_D, DS_{IN} \rangle$*, is*

$$DS_P = \begin{cases} AS_A \textit{ if } \text{Effect} = \text{Permit} \\ \emptyset \quad \textit{otherwise} \end{cases}$$

$$DS_D = \begin{cases} AS_A \textit{ if } \text{Effect} = \text{Deny} \\ \emptyset \quad \textit{otherwise} \end{cases}$$

$$DS_{IN} = AS_{IN}$$

In order to obtain the decision space of the root policy element, the decision space of child policy elements have to be recursively combined in a bottom-up fashion according to specified combining algorithms. As noted in Section 2.1 some combining algorithms use an extended decision set in which the *Indeterminate* space is subdivided into three parts. For these we extend the decision space accordingly. Here, we show the decision space of a policy with respect to deny-overrides as an example and provide the encoding of other common algorithms in the Appendix B. Let $\langle DS_P^{p_1}, DS_D^{p_1}, DS_{IN(P)}^{p_1}, DS_{IN(D)}^{p_1}, DS_{IN(PD)}^{p_1} \rangle$ and $\langle DS_P^{p_2}, DS_D^{p_2}, DS_{IN(P)}^{p_2}, DS_{IN(D)}^{p_2}, DS_{IN(PD)}^{p_2} \rangle$ be the (extended) decision spaces of policy elements p_1 and p_2, respectively. We are interested in the decision space $\langle DS_P^p, DS_D^p, DS_{IN}^p \rangle$ of a policy p which combines policy elements p_1 and p_2 using deny-overrides.

The decision spaces induced by deny-overrides can be defined as follows:

$$DS_D^p = DS_D^{p_1} \cup DS_D^{p_2}$$

$$DS_{IN(PD)}^p = \{(DS_{IN(PD)}^{p_1} \cup DS_{IN(PD)}^{p_2}) \cup [DS_{IN(D)}^{p_1} \cap (DS_{IN(P)}^{p_2} \cup DS_P^{p_2})] \cup [DS_{IN(D)}^{p_2} \cap (DS_{IN(P)}^{p_1} \cup DS_P^{p_1})]\} \setminus DS_D^p$$

$$DS_{IN(D)}^p = (DS_{IN(D)}^{p_1} \cup DS_{IN(D)}^{p_2}) \setminus (DS_D^p \cup DS_{IN(PD)}^p)$$

$$DS_P^p = (DS_P^{p_1} \cup DS_P^{p_2}) \setminus (DS_D^p \cup DS_{IN(PD)}^p \cup DS_{IN(D)}^p)$$

$$DS_{IN(P)}^p = (DS_{IN(P)}^{p_1} \cup DS_{IN(P)}^{p_2}) \setminus (DS_D^p \cup DS_{IN(PD)}^p \cup DS_{IN(D)}^p \cup DS_P^p)$$

Intuitively, the representation above defines the priorities between decision spaces. The *Deny* space of the parent policy element is the union of the *Deny* space of child policy elements, i.e. the former evaluates to *Deny* if at least one child policy element evaluates to *Deny*. Then, *Indeterminate*$\{PD\}$ has priority over *Indeterminate*$\{D\}$; in turn *Indeterminate*$\{D\}$ has priority over *Permit*, which has priority over *Indeterminate*$\{P\}$. The overall *Indeterminate* space can be obtained as the union of the three *Indeterminate* spaces, i.e. $DS^p_{IN} = DS^p_{IN(PD)} \cup DS^p_{IN(D)} \cup DS^p_{IN(P)}$.

Example 3 *Consider the policy in Example 1 and the applicability space of the rules forming it in Example 2. Decision space of rule* r_i $\langle DS^{r_i}_P, DS^{r_i}_D, DS^{r_i}_{IN}\rangle$ *is*
$r_1 : \langle \emptyset, ac_0 \cap ac_1 \cap ac_5, (ac_8 \cup ac_9 \cup ac_{10}) \cap (ac_0 \cap ac_1)\rangle$
$r_2 : \langle \emptyset, ac_0 \cap ac_1 \cap (ac_2 \cup ac_3 \cup ac_4), (ac_6 \cup ac_7) \cap (ac_0 \cap ac_1)\rangle$
$r_3 : \langle ac_0 \cap ac_1, \emptyset, \emptyset\rangle$

The decision space of the overall policy $\langle DS_P, DS_D, DS_{IN}\rangle$ *can be obtained by combining the decision space of the rules as shown above:*

$$DS^p_D = ac_0 \cap ac_1 \cap \overline{(ac_5 \cup ac_2 \cup ac_3 \cup ac_4)}$$
$$DS^p_P = ac_0 \cap ac_1 \cap \overline{(ac_2 \cup ac_3 \cup ac_4 \cup ac_5 \cup ac_6 \cup ac_7 \cup\ ac_8 \cup ac_9 \cup ac_{10})}$$
$$DS^p_{IN} = ac_0 \cap ac_1 \cap (ac_6 \cup ac_7 \cup ac_8 \cup ac_9 \cup ac_{10}) \cap \overline{(ac_5 \cup ac_2 \cup ac_3 \cup ac_4)}$$

where notation $\overline{S}$ *is used to denote the complement of set* S.

3.2 Policies as SMT Formulas

As shown above, the applicability space and, thus, the decision space of policies can be expressed as many-sorted first-order predicates over attributes they contain. To this end, we encode the decision space of a policy using SMT formulas.

Definition 3 *Given an XACML policy* p *and a background theory* $\mathcal{T}$*, the representation of* p *in SMT is a tuple* $\langle \mathcal{F}_P, \mathcal{F}_D, \mathcal{F}_{IN}\rangle$ *where* $\mathcal{F}_P$*,* $\mathcal{F}_D$ *and* $\mathcal{F}_{IN}$ *are many sorted first-order formulas encoding* Permit*,* Deny*,* Indeterminate *decision spaces of* p *respectively with some of their terms interpreted in* $\mathcal{T}$*.*

When talking about deciding satisfiability of a policy p in SMT, we refer to $\mathcal{T}$-satisfiability of the formulas $\mathcal{F}_P$, $\mathcal{F}_D$ and $\mathcal{F}_{IN}$. Since decision spaces DS_P, DS_D and DS_{IN} are pair-wise disjoint, their satisfiability is mutually exclusive, that is if $\mathcal{F}_P$ is $\mathcal{T}$-satisfiable then $\mathcal{F}_D$ and $\mathcal{F}_{IN}$ are not $\mathcal{T}$-satisfiable and so on.

The background theories needed for the analysis of a policy are determined from the policy's applicability constraints. In order to do this, we map classes of common XACML functions to certain background theories that can be used to encode the applicability constraints constructed from them.

Most of the logical functions of XACML (i.e., *or*, *and*, *not*) do not require any specific background theory. Some applicability constraints involving equality predicates of attributes with finite domains can be modeled by the theory of enumerated data types in which attribute values are represented as 0-ary function symbols within an appropriate signature Σ. Other constraints involving equality predicates require the theory of equality with uninterpreted functions. This theory does not impose any constraint on the way the symbols in the signature are

interpreted. Thus, the predicates that are not supported by any theory can be left uninterpreted and analyzed using the theory of "uninterpreted functions". The theory of equality with uninterpreted functions can be used to support XACML functions for which a dedicated theory is not available such as XPath-based functions. Constraints defined using arithmetic and numeric comparison functions (e.g., $ac_3, \ldots, ac_5$ in Example 2) require the theory of linear arithmetic. Applicability constraints defined over strings, bag and sets may require dedicated theories. For instance, constraints defined using comparison functions over strings and string conversion functions can be modeled with the theory of strings [32]; constraints defined over bag and set functions (e.g., $ac_6, \ldots, ac_{10}$) can be modeled with the theories of arrays [19] and cardinality constraints on sets [29].

Finally, observe that a background theory can be a combination of different theories as it is the case of Example 2 in which the cost attribute is defined as a user-defined function $f(\mathsf{value})$ over the attribute value. By abstracting the actual details of f, the applicability constraint ac_5 can be represented as $(\mathsf{credit} < f(\mathsf{value}) + \mathsf{value})$ and interpreted within a combined background theory of linear arithmetic and uninterpreted functions. See the Appendix C for a summary of the mapping between common XACML functions and background theories, and the respective computational complexity.

4 XACML Policy Analysis

The previous section describes an encoding of XACML policies as SMT formulas. In this section we use this encoding to represent *policies analysis problems*, i.e. for a collection of policies checking various properties expressed in so called *queries*. We first introduce the query language and then give example query formulas for different policy properties from the literature.

Definition 4 *Let $\langle Att, Dom\rangle$ be the access control scheme and $\mathcal{T}$ a background theory with signature Σ. A* policy analysis problem *is a tuple $\langle Q, (p_1, \ldots, p_n)\rangle$ where $p_1, \ldots, p_n$ are policies expressed in SMT with respect to $\mathcal{T}$ and Q is a* (policy) query*. A query Q is a formula of the form*

$$\begin{aligned} Q = {} & P_i \mid D_i \mid IN_i \mid g(t_1, \ldots, t_k) \mid \neg Q \mid Q_1 \wedge Q_2 \mid Q_1 \vee Q_2 \mid Q_1 \rightarrow Q_2 \\ & \mid (\forall x : \sigma\ Q) \mid (\exists x : \sigma\ Q) \mid Q[x/t] \mid \nu x.P \mid Q\langle a_1 = v_1, \ldots, a_n = v_n\rangle \end{aligned}$$

where P_i, D_i and IN_i (for $i = 1, \ldots, n$) are new symbols representing the Permit*,* Deny *and* Indeterminate *spaces of policy p_i (they thus represent $\mathcal{F}_P^{p_i}$, $\mathcal{F}_D^{p_i}$ and $\mathcal{F}_{IN}^{p_i}$ respectively, see also query semantics below), g is a Σ-atom over terms $t_1, \ldots, t_k$ such that each term t is either a variable denoting attributes from Att or built using function symbols in Σ^F, and the logical operators are defined as usual where Q_1 and Q_2 are also queries. $Q[x/t]$ represents the substitution of variable x with term t in Q, $\nu x.Q$ represents the restriction of a variable x in Q (i.e., $\nu x.Q \equiv Q[x/y]$ with y a fresh variable), $Q\langle a_1 = v_1, \ldots, a_n = v_n\rangle$ represents the instantiation of a policy with a request and is logically equivalent to $Q \wedge v_1 \in a_1 \wedge \ldots \wedge v_n \in a_n$.*

Note that construct $\nu x.Q$ is used to restrict the scope of the substitution of a variable x to a subformula Q of the query. This allows us to encode properties comparing a number of policies, in which some policies are instantiated with a request while other policies are instantiated with a different request (see below for examples of such properties).

The basic query P_i encodes (inclusion in) the *Permit* space of policy p_i; it is satisfiable if any request is permitted by p_i. Similarly, D_i and IN_i represent the *Deny* and *Indeterminate* spaces of p_i respectively. Predicates on terms such as $\mathbf{Alice} \in \mathsf{subject\text{-}id}$, $\forall v \in \mathsf{current\text{-}time}\ x < 18{:}00$ etc., are used to instantiate the subject or the time of the query. The predicates can also capture relations between different policies (see examples below).

Example 4 *Let p_1, p_2 be two policies, and $\langle P_1, D_1, IN_1\rangle$ and $\langle P_2, D_2, IN_2\rangle$ their SMT representation, respectively. Below we present some example queries.*

- $(P_1 \to P_2)$*: any request permitted by p_1 is also permitted by p_2.*
- $\nu\ \mathsf{subject\text{-}id}.(P_1\langle \mathsf{subject\text{-}id} = \mathbf{Alice}\rangle) \wedge \nu\ \mathsf{subject\text{-}id}.(\mathsf{D}_1\langle \mathsf{subject\text{-}id} = \mathbf{Bob}\rangle)$*: some request is permitted by p_1 for Alice but denied for Bob.*
- $(P_1 \wedge D_2)\langle \mathsf{subject\text{-}id} = \mathbf{Alice}\rangle$*: some request of Alice is permitted by p_1 but denied by p_2.*
- $P_1\langle \mathsf{subject\text{-}id} = \mathbf{Alice}, \mathsf{resource\text{-}type} = \mathbf{transaction}, \mathsf{action\text{-}id} = \mathbf{create}\rangle$*: policy p_1 allows Alice to create a transaction.*

Definition 5 *Let $\langle Q, (p_1, \ldots, p_n)\rangle$ be a policy analysis problem, $\mathcal{T}$ a background theory with signature Σ, and $\mathcal{M}$ the structure for signature Σ. Let $\langle \mathcal{F}_P^{p_i}, \mathcal{F}_D^{p_i}, \mathcal{F}_{IN}^{p_i}\rangle$ be the encoding of policy p_i in SMT with some or all terms interpreted in $\mathcal{T}$. We say that a valuation ϕ is a solution to $\langle Q, (p_1, \ldots, p_n)\rangle$ if*

$$\mathcal{M}, \phi \models Q[\mathcal{F}_P^{p_i}/P_i, \mathcal{F}_D^{p_i}/D_i, \mathcal{F}_{IN}^{p_i}/IN_i]_{i=1,\ldots,n}$$

We say that $\langle Q, (p_1, \ldots, p_n)\rangle$ is satisfiable with respect to $\mathcal{T}$ if the policy analysis problem has a solution. Otherwise, we say that it is unsatisfiable.

In the remainder of this section, we demonstrate that our framework can model various types of policy properties proposed in the literature.

Policy Refinement and Subsumption Organizations often need to update their security policies to comply with new regulations or to adapt changes in their business model. Nonetheless, they might have to ensure that the new policies preserve (refine) the intention of the original policies. Different definitions of policy refinement have been proposed in the literature. Backes et al. [4] propose a notion of policy refinement based on the idea that *"one policy refines another if using the first policy automatically also fulfills the second policy"*. Intuitively, a policy refines another policy if whenever the latter returns *Permit* (or *Deny*) the first policy returns the same decision. This can be formalized in our framework as follows. Let p_1, p_2 be two policies with decision space $\langle P_1, D_1, IN_1\rangle$ and $\langle P_2, D_2, IN_2\rangle$ respectively. Policy p_2 is a refinement of p_1 iff

$$\forall\langle a_1{=}v_1, \ldots, a_n{=}v_n\rangle \in \mathcal{R}\ ((P_1 \to P_2) \wedge (D_1 \to D_2))\langle a_1{=}v_1, \ldots, a_n{=}v_n\rangle \quad (1)$$

Hughes and Bultan [16] present a stronger notion of policy refinement called policy subsumption. In addition to constraining *Permit* and *Deny* spaces as in refinement, subsumption also imposes constraints on the *Indeterminate* space. Formally, policy p_1 subsumes policy p_2 iff

$$\begin{aligned} &\forall\langle a_1 = v_1, \ldots, a_n = v_n\rangle \in \mathcal{R} \\ &\quad ((P_1 \rightarrow P_2) \wedge (D_1 \rightarrow D_2) \wedge (IN_1 \rightarrow IN_2))\langle a_1 = v_1, \ldots, a_n = v_n\rangle \end{aligned} \tag{2}$$

Note that our framework is general enough to express other notions of policy refinement, for instance imposing constraints only on the *Permit* space or on the *Deny* space. In the next example, we demonstrate the notion of policy refinement presented in [4] with respect to background theory *linear arithmetic*.

Example 5 *Consider the XACML policy in Example 1. Suppose that the policy is updated by omitting the cost of the transaction in rule* r_1*:*

$$r_1'[Deny] : \mathsf{value} > \mathsf{credit}$$

We want to check whether the new policy is a refinement of the original policy. It is easy to verify that (1) does not hold if the cost of the transaction is higher than the credit minus the value of the service. Therefore, the new policy is not a refinement of the original policy.

Change-impact Change-impact analysis [11] aims to analyze the impact of changes to policies. Intuitively, change-impact analysis is the counterpart of policy refinement, in which the goal is to extract the differences between two policies. Differently from policy refinement, changes of the *NotApplicable* space should also be considered in change-impact analysis. Let p_1, p_2 be two policies with decision space $\langle P_1, D_1, IN_1\rangle$ and $\langle P_2, D_2, IN_2\rangle$ respectively. We are interested in finding the access requests for which the decisions returned by p_1 and p_2 are different. This policy analysis problem can be formalized in our framework as

$$\begin{aligned} &(P_1 \rightarrow \neg P_2) \vee (D_1 \rightarrow \neg D_2) \vee (IN_1 \rightarrow \neg IN_2) \\ &\vee(\neg(P_1 \vee D_1 \vee IN_1) \rightarrow (P_2 \vee D_2 \vee IN_2)) \end{aligned} \tag{3}$$

where $\neg(P_1 \vee D_1 \vee IN_1)$ represents the *NotApplicable* space.

Attribute Hiding An attribute hiding attack is a situation in which a user is able to obtain a more favorable authorization decision by hiding some of her attributes [8]. Attribute hiding attack is a threat exploiting the non-monotonicity of access control systems such as XACML. Differently from the previous policy properties that can be expressed solely in terms of *Permit*, *Deny* and *Indeterminate* spaces of the policies, attribute hiding is about changing the request: a request that is previously denied is permitted by hiding some attributes or attribute-value pairs. In particular, we call *partial attribute hiding* attack the situation in which a user hides a single attribute-value pair. Let $req = \langle a_1 = v_1, \ldots, a_n = v_n\rangle$ be

a request denied by a policy p (i.e. a solution of D_p), and $a_i = v_i$ an attribute-value pair occurring in req ($1 \leq i \leq n+1$). A policy is vulnerable to partial attribute hiding attack if the request obtained by suppressing $a_i = v_i$ from req is permitted by p (i.e. a solution of P_p). The property representing the absence of partial attribute hiding attack can be encoded as follows:

$$\nu a.(D_p\langle a = v\rangle) \rightarrow \neg P_p \tag{4}$$

where we use restriction to ensure that the request is only applied to the left part of the formula. A more generalized version of attribute hiding attack is *general attribute hiding* where a user completely suppresses information about one attribute. The property representing the absence of general attribute hiding attack can be encoded as follows:

$$\nu a.(D_p\langle a = v_1, \ldots, a = v_n\rangle) \rightarrow \neg P_p \tag{5}$$

We use an example policy from [8] to discuss the analysis of attribute hiding.

Example 6 *Consider two competing companies, A and B. To protect confidential information from competitors, company A defines the following policy:*

$$\begin{aligned} p[\mathsf{dov}] &: \mathbf{true} \\ r_1[Deny] &: \mathsf{confidential} = true \wedge \mathsf{employer} = B \\ r_2[Permit] &: \mathbf{true} \end{aligned}$$

*The first rule (r_1) of the policy denies employees of company B to access confidential information while the second rule (r_2) grants access to every requests. The two rules are combined using deny-overrides combining algorithm (*dov*). Now consider the following access requests:*

$$\begin{aligned} req_1 =& \langle \mathsf{employer} = A, \mathsf{confidential} = true\rangle \\ req_2 =& \langle \mathsf{employer} = A, \mathsf{employer} = B, \mathsf{confidential} = true\rangle \\ req_3 =& \langle \mathsf{confidential} = true\rangle \end{aligned}$$

Rule r_1 is only applicable to request req_2 and thus the request is denied. Rule r_2 is applicable to the remaining requests and thus access is granted for requests req_1 and req_3. However, if the subject can hide some information from the request, for instance, reducing req_2 to req_1 by suppressing element $\mathsf{employer} = B$ *from the request (partial attribute hiding) or to req_3 by suppressing attribute* employer *from the request (general attribute hiding), then she would be allowed to access confidential information leading to a violation of the conflict of interest requirement. Note that we assume that attribute* confidential *is under the control of the system and cannot be hidden by the user.*

Scenario-finding Scenario finding queries [11, 23] aim to find attribute assignments that represent scenarios in which a sought behavior occurs. They are especially useful to obtain request instances of certain decision types (e.g., *permit*)

which are otherwise difficult to obtain manually. Examples of scenario finding queries include checking whether a policy ever permits (some) users to perform certain actions or denies certain actions under given circumstances. Scenario finding can also be used to check whether a policy is compliant with well-known security principles. For instance, a XACML policy implementing role-based access control can be checked for the separation of duty principle or a XACML policy implementing Chinese Wall policy can be checked if it correctly implements conflict of interest classes.

Scenario finding does not have a fixed form of encoding as the previous properties since it is formulated by the user according to selected decision space.

Example 7 *In the context of our example policy, a policy author may want to check whether the policy permits any access request before* 18:00 *on* Saturday. *We can encode this query as follows:*

$$P\langle \mathit{current\text{-}day} = \mathit{Saturday} \rangle \wedge \mathsf{time} < 18{:}00$$

Many types of scenario finding queries can be formulated and analyzed within existing XACML analysis tools. However, most of these tools leave non-Boolean functions (i.e., Σ-terms of form $f(t_1, \ldots, t_n)$) uninterpreted. In contrast, SMT enables to reason on those attributes using a suitable underlying background theory. For instance, an SMT solver can find an assignment for an attribute "age" that satisfies the a linear arithmetic predicate $\mathsf{age} < 18$.

5 Evaluation

In this section we evaluate our SMT-based policy analysis framework by means of a prototype implementation. In the evaluation we use two sets of experiments, one comparing our SMT-based solution to SAT-based techniques and one showing our prototype can be used on realistic policies. Our experimental testbed consists of a 64-bit (virtual) machine with 16GB of RAM and 3.40GHz quad-core CPU running Ubuntu.

5.1 Prototype Implementation

To support the analysis of XACML policies described in the previous section, we have implemented *X2S* [30], a formal policy analysis tool. *X2S* employs Z3 [22], an SMT-LIB v2 compliant tool that supports efficient reasoning in a wide range of background theories, as the underlying solver. *X2S* accepts both XACML v2 and v3 policies and supports a large fraction of standard XACML functions. It consists of two main components. The first component, the *SMT Translator*, first translates XACML policies provided by the user into an SMT formula using the encoding presented in Section 3. Next the user is prompted to enter a query expressed in the language defined in Section 4 which is also translated and added

to SMT specification. The second component, the *Report Generator*, presents the results of the analysis by providing an interface to the SMT solver.

Our prototype can, by using Z3, enumerate models as required for queries such as *change impact*. However, there may be infinitely many models satisfying a formula. To help alleviate this problem, we try to avoid models that do not "significantly" differ from those already considered with respect to the assignments of the attributes. In particular, we do this in the treatment of arithmetic expressions. For instance, if a numerical attribute *att* has been assigned the value 5 because of a constraint requiring it to be strictly greater than 4, then we will avoid considering the assignments that assign *att* the increasing values 6, 7, and so on.

5.2 Experiments 1: SAT vs. SMT

Consider a user wanting to validate and possibly update a set of policies collected over time and from different contributors. For example, a building manager wants to verify the policy governing the access to a certain building in which right to enter depend on the current time and date and/or membership of a group; or a bank manager wants to verify the bank policy for transfers which depend on the balance of accounts, size of the transfer, etc. The main advantage of our SMT approach over a SAT based solution is that it allows direct reasoning with non-Boolean values. For example, one can use the background theories for basic sets (i.e., the theory of arrays) and linear arithmetic (LAI). To perform this analysis in SAT one has to encode everything in Boolean values. With some limitations we can encode LAI constraints in SAT using order encoding [25] where each expression of the form $x \leq c$ is represented by a different Boolean variable. Membership expressions in the set theory can be encoded in SAT using a similar approach where the relation between a variable and a value from its domain is represented with a different Boolean variable for each value.

Ideally the user's validation tool would be able to give real-time feedback on their edits, or at the very least, respond promptly to a validation query. When analyzing with SAT the user will have to compromise the level of granularity of the analysis; for example instead of the time only distinguishing 'morning' from 'afternoon' or hour of the day. Choosing what granularity is suitable for what attribute is a difficult, laborious and error-prone task requiring the user to closely investigate all constraints. A too low granularity may lead to missing errors in the policies. Yet, the more fine grained the analysis is, the larger the SAT encoding. Our experiment below confirm that increasing the granularity quickly become very costly performance wise. Our SMT-based approach does not needed to restrict the granularity.

To illustrate the effect of granularity on the analysis we distinguish course grained analysis using a 'small' domain (e.g., morning/afternoon for time, and day of the week for date), an analysis with some detail through a 'medium' (M) size domain (e.g., minutes in an hour, days in a month) and a detailed analysis using a 'large' (L) domain (e.g., minute in the day, day of the year, actual balance up-to a million). We analyze policies and properties from the examples

Table 1: Evaluation Results of Example Properties with SAT vs SMT Encoding

		#Vars			**Memory(MB)**							**Time(s)**						
		SAT		SMT	Z3-SAT		zchaff		lingeling		SMT	Z3-SAT		zchaff		lingeling		SMT
P	**Q**	M	L		M	L	M	L	M	L		M	L	M	L	M	L	
PR	$\neg\mathcal{F}$	591	2191	12	84	459	99	340	23	555	0.3	1.6	99.7	~0	3.3	20.5	>100	~0
PS	$\neg\mathcal{F}$	909	2509	12	303	240	377	1159	82	2054	0.3	3.9	6.1	~0	12.4	65.5	>100	~0
CI	$\mathcal{F}$	1409	3009	12	88	231	650	1087	133	1513	0.5	0.3	9.1	~0	19.1	36.5	45.5	~0
P-AH	$\neg\mathcal{F}$	24	15	3	0.1	0.1	0.1	0.1	~0	~0	0.3	~0	~0	~0	~0	~0	~0	~0
G-AH	$\neg\mathcal{F}$	12	12	4	~0	~0	0.1	0.1	~0	~0	0.3	~0	~0	~0	~0	~0	~0	~0
SF	$\mathcal{F}$	511	1718	12	13	328	92	409	14	356	0.3	~0	1.2	~0	13.4	0.2	7.3	~0

in Section 4. We check policy refinement (PR), policy subsumption (PS), change-impact (CI) analysis, both partial (P-AH) and global (G-AH) Attribute Hiding, and finally scenario finding (SF). We analyze each with our prototype and three different SAT solvers; zchaff [21], lingeling [6] and Z3 itself to obtain a fair comparison as certain solvers are optimized for certain types of problems. We use size 10 to represent small domains, 100 for medium domains and 500 for large domains (they may need to be much larger but this size already shows the clear advantage of our SMT solution). Note that we aim at a comparison in orders of magnitude rather than an in-depth and comprehensive performance analysis. For small domains all solutions are able to complete the analysis quickly with limited resources; they are fast enough for real-time feedback during editing. For the medium and large domains the results are provided in Table 1. The first column specifies the property (**P**) analyzed. The second column (**Q**) gives the class of formula used; finding a counter-example ($\neg\mathcal{F}$) or a satisfying assignment ($\mathcal{F}$). The other columns present the results in terms of number of variables used in the encoding, memory allocation[4] and required computation time for the SAT solvers and SMT.

Compared to the number of many-sorted first order variables in SMT encoding, the number of Boolean variables in SAT encoding is quite large due to the mapping of non-Boolean domains to Boolean variables. For instance, the SMT encoding of the policy query for verifying policy refinement requires 12 variables. These variables are used to specify the attributes defined in the policy as well as the Boolean variables representing one-and-only constraints on the arithmetic variables (Example 1). In contrast, 591 Boolean variables are needed to encode the same policy query in SAT when a medium size domain is considered. The memory allocated by the SMT solver needed in analyzing the example policies was always less than 1MB for all properties while SAT solver requires several orders of magnitude more memory. The time necessary to prove (or disprove) that the property holds was negligible (~10ms) for all SMT cases. Analysis with SAT solvers performs far worse with the growth of the domain size as can be noted from the table. For instance, for scenario finding analysis with a large do-

[4] We used a memory profiler for measuring the memory usage.

main the best performing SAT solver (Z3) took ∼1.2s which is several orders of magnitude slower than the analysis with SMT (which took 7ms). The exception is the case of attribute hiding analysis where the SAT solvers offer performance similar to SMT. This is expected since our example policy for attribute hiding does not include complicated predicates and the available predicates can be easily represented in propositional logic. Note that in our experiments for the case of change-impact analysis, we obtained only one model since we prune the uninteresting assignments of arithmetic variables (i.e. value, credit and cost). Finally, we also observe a performance variation between different SAT solvers. We believe this is due to the fact that certain solvers are better tailored to certain types of problems.

In conclusion, even with these relatively simple policies, performance quickly becomes impractical using SAT based solvers while the SMT approach could even be used for real-time feedback while editing a policy. In the next section we check our solution with some more complex and realistic policies.

5.3 Experiments 2: Real-world Policies

In this second set of experiments, we analyze four realistic policies with our prototype in order to obtain insights about its performance in real-world settings. The policy GradeMan is a simplified version of the access control policy used to regulate access to grades at Brown university and the Continue-a policy is used to manage a conference management system. Both policies are from [11] and consist mainly of string equality predicates. IN4STARS is an in-house policy defined in the context of a project on intelligence interoperability. It contains various user-defined functions that are used to determine the privileges of users according to their clearance. All these three policies are XACML v2 policies. Our final test policy, KMarket, is a sample policy to manage authorizations in an on-line trading application from [1]. It contains simple arithmetic operations such as less-than and is written in XACML v3.

We performed policy refinement, subsumption and change-impact analysis by modifying the value of a single, randomly chosen attribute in the original policy. The number of models has been limited to 100 during change-impact analysis. In the scenario finding experiments, we look for an assignment of attributes (i.e. model) that is permitted by the input policy. Our findings are summarized in Table 2 in which we report the characteristics of policies (e.g., the number of policy elements in the XACML policy) and the time taken by our prototype to answer queries.

Analyzing the policies included in our experiments takes less than 100ms for all properties except Change-impact which makes feedback during policy editing feasible. Change-impact analysis, however, brings the time up to 3s as it requires the enumeration of models in the SMT formula. Another important observation in the experiments is efficiency of dealing with expressions with non-Boolean attributes; we have not observed a significant performance difference between the analysis of KMarket which contains linear arithmetic expressions and GradeMan which consists of very simple expressions. Finally, the result of

Table 2: Evaluation Results for Real-world Policies

Policy	#PSet	#Policy	#Rule	Time(ms)					
				PR	PS	CI	P-AH	G-AH	SF
IN4STARS	3	4	11	24	28	1717	7	7	10
KMarket	1	3	12	36	12	2525	13	12	10
GradeMan	11	5	5	40	30	2424	10	9	17
Continue-a	111	266	298	91	87	2929	33	21	43

Continue-a analysis (a policy with 300 rules) indicates that the time needed for analysis with SMT of larger policies increases but not necessary as quickly as the policy grows. This result is not surprising since the analysis of a policy with our approach not only depends on the size of the policy but also the type of expressions contained in them.

We believe the experimental results of this and the previous section demonstrate that our approach can be used in practice to analyze realistic policies at a more fine-grained level than the one permitted by the use of SAT solvers with no significant performance penalty.

6 Related Work

When XACML policies grow in number and size, or are updated to address new security requirements, it is difficult to verify their correctness due to XACML's rich and verbose syntax. To assist policy authors in the analysis of XACML policies, several policy analysis tools have been proposed. One of the most prominent tools for policy analysis is Margrave [11]. Margrave uses multi-terminal binary decision diagrams (MTBDDs) as the underlying representation of XACML policies. The nodes of an MTBDD represent Boolean variables encoding the attribute-values pairs in the policy. The terminal nodes represent the possible decisions (i.e., *NotApplicable*, *Permit* or *Deny*). Given an assignment of Boolean values to the variables, a path from the root to a terminal node according to the variable values indicates the result of the policy under that assignment. Margrave uses MTBDDs to supports two types of analysis: policy querying, which analyzes access requests evaluated to a certain decision, and change-impact analysis, which is used to compare policies. Another policy analysis tool that employs BDDs for the encoding of XACML policies is XAnalyzer [15]. XAnalyzer uses a policy-based segmentation technique to detect and resolve policy anomalies such as redundancy and conflicts. Compared to our approach, BDD-based approaches allow the verification of XACML policies against a limited range of properties. In addition, these approaches encode only a fragment of XACML: they can only handle simple constraints; complex constraints cannot be expressed, not even as uninterpreted Boolean variables [16].

An alternative to Margrave, and in general to BDD-based approaches, is presented in [16] where policies and properties are encoded as propositional formulas

and analyzed using a SAT solver. However, SAT solvers cannot handle non-Boolean variables; most XACML functions are thus left uninterpreted limiting the capability of the analysis. EXAM [20] combines the use of SAT solvers and MTBDD to reason on various policy properties. In particular, EXAM supports three classes of queries: *metadata* (e.g., policy creation date), *content* (e.g., number of rules) and *effect* (e.g., evaluation of certain requests). Policies and queries are expressed as Boolean formulas. These formulas are converted to MTBDDs and then combined into a single MTBDD for analysis.

Other formalisms have also been used for the analysis of XACML policies. For instance, Kolovski et al. [17] use description logic (DL) to formalize XACML policies and employs off-the-shelf DL reasoners for policy analysis. The use of DL reasoners enables the analysis on a wide subset of XACML in a more expressive manner but also hinders the performance. Ramli et al. [27] and Ahn et al. [2] present a formulation of policy analysis problems similar to ours in answer set programming (ASP). However, these approaches have drawbacks due to intrinsic limitations of ASP. Unlike SMT, ASP does not support quantifiers, and cannot easily express constraints such as linear arithmetic. Indeed, in ASP the grounding (i.e., instantiation of variables with values) of linear arithmetic constraints either yield very large number of clauses (integers) or is not supported (reals).

In summary, the approaches discussed above lack the inherent benefits of SMT: either background theories are not supported so that the attributes involved in most XACML functions cannot be analyzed at a finer level, or the performance of analysis deteriorates very quickly.

While the use of SMT for the analysis of XACML policies is new to our knowledge, there are few recent proposals that exploit SMT solvers for the analysis of policies specified in different access control models. For instance, Armando et al. [3] use SMT to detect conflicts and redundancies in RBAC. Here, rules specifying constraints on the assignment/activation of roles are encoded as SMT formulas with certain background theories such as enumerated data types and linear arithmetic for reals/integers. Another example is the work in [28] which proposes safety analysis of UCON policies with SMT. Although these proposals show the potentiality of SMT for policy analysis, the policy specifications considered in such proposals are rather simple. In this work we make an additional step by showing that SMT is able to deal with real world XACML policies.

Next to proposals for policy analysis, we can find proposals aiming at the generation of XACML policies, where access control policies expressed in some formalism are first analyzed and then translated to XACML. For instance, Zhang et al. [31] propose RW (Read and Write), a formalism based on propositional logic, for the specification of access control policies. Based on this formalism, they employ symbolic model-checking to verify properties against a policy model expressed in RW. Policy model specified in the RW language are then translated into XACML. Although this approach may simplify the design of new policies, it is not suitable for the analysis of existing XACML policies.

7 Conclusions

In this paper, we presented an SMT-based analysis framework for policies specified in XACML. The use of SMT does not only enable wider coverage of XACML compared to existing analysis tools but also presents significant performance gains in terms of allocated memory and computational time. As demonstrated in the paper, several security policy properties found in the literature can be easily encoded and checked within our framework. In our prototype, we use various background theories to encode a large fraction of XACML functions, allowing a fine-grained analysis of XACML policies. SMT function symbols encoding XACML functions for which a specific background theory is not available (e.g., XPath-based and regular-expression-based functions) are left uninterpreted. With the development of new background theories, policy analysis problems using those predicates can be represented and solved efficiently. Our experiments show that our framework enables efficient policy analysis and can be used in practice. As future work, we plan to extend the performance analysis of our prototype against a larger set of real-world policies.

Acknowledgments This work has been partially funded by the EDA project IN4STARS2.0, the EU FP7 project AU2EU, the ARTEMIS project ACCUS, and the Dutch national program COMMIT under the THeCS project.

References

1. Balana: Open source xacml 3.0 implementation (Jan 2013), `http://xacmlinfo.org/category/balana/`
2. Ahn, G.J., Hu, H., Lee, J., Meng, Y.: Representing and reasoning about web access control policies. In: COMPSAC. pp. 137–146 (2010)
3. Armando, A., Ranise, S.: Automated and efficient analysis of role-based access control with attributes. In: DBSec. pp. 25–40 (2012)
4. Backes, M., Karjoth, G., Bagga, W., Schunter, M.: Efficient comparison of enterprise privacy policies. In: SAC. pp. 375–382 (2004)
5. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, pp. 825–885. IOS Press (2008)
6. Biere, A.: Lingeling essentials, A tutorial on design and implementation aspects of the the SAT solver lingeling. In: POS. p. 88 (2014)
7. Bjørner, N.: Linear Quantifier Elimination as an Abstract Decision Procedure. In: IJCAR. pp. 316–330 (2010)
8. Crampton, J., Morisset, C.: PTaCL: A Language for Attribute-Based Access Control in Open Systems. In: POST. pp. 390–409 (2012)
9. Enderton, H.B.: A Mathematical Introduction to Logic. Academic Press (1972)
10. Fischer, M.J., Rabin, M.O.: Super-exponential complexity of presburger arithmetic. In: Quantifier Elimination and Cylindrical Algebraic Decomposition, pp. 122–135. Texts and Monographs in Symbolic Computation, Springer (1998)
11. Fisler, K., Krishnamurthi, S., Meyerovich, L.A., Tschantz, M.C.: Verification and change-impact analysis of access-control policies. In: ICSE. pp. 196–205 (2005)
12. Ge, Y., de Moura, L.: Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories. In: CAV. LNCS, vol. 5643, pp. 306–320. Springer (2009)

13. Gomes, C.P., Kautz, H., Sabharwal, A., Selman, B.: Satisfiability Solvers. In: Handbook of Knowledge Representation, Foundations of Artificial Intelligence, vol. 3, pp. 89–134. Elsevier (2008)
14. Halpern, J.Y.: Presburger Arithmetic with Unary Predicates is Π_1^1-complete. Journal of Symbolic Logic 56, 56–2 (1991)
15. Hu, H., Ahn, G.J., Kulkarni, K.: Discovery and Resolution of Anomalies in Web Access Control Policies. TDSC 10(6), 341–354 (2013)
16. Hughes, G., Bultan, T.: Automated verification of access control policies using a SAT solver. STTT 10(6), 503–520 (2008)
17. Kolovski, V., Hendler, J.A., Parsia, B.: Analyzing web access control policies. In: WWW. pp. 677–686 (2007)
18. Kroening, D., Strichman, O.: A framework for Satisfiability Modulo Theories. Formal Asp. Comput. 21(5), 485–494 (2009)
19. Kröning, D., Weissenbacher, G.: A Proposal for a Theory of Finite Sets, Lists, and Maps for the SMT-Lib Standard. In: Pro. International Workshop on Satisfiability Modulo Theories (2009)
20. Lin, D., Rao, P., Bertino, E., Li, N., Lobo, J.: Exam: a comprehensive environment for the analysis of access control policies. Int. J. Inf. Sec. 9(4), 253–273 (2010)
21. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC. pp. 530–535 (2001)
22. de Moura, L.M., Bjørner, N.: Z3: An Efficient SMT Solver. In: TACAS. pp. 337–340 (2008)
23. Nelson, T.: First-order Models For Configuration Analysis. Ph.D. thesis, Worcester Polytechnic Institute (2013)
24. OASIS XACML Technical Committee: eXtensible Access Control Markup Language (XACML) (2013)
25. Petke, J., Jeavons, P.: The Order Encoding: From Tractable CSP to Tractable SAT. In: SAT. pp. 371–372 (2011)
26. Pratt, V.R.: Two easy theories whose combination is hard. Tech. rep., MIT (1977)
27. Ramli, C.D.P.K., Nielson, H.R., Nielson, F.: XACML 3.0 in Answer Set Programming. In: LOPSTR (2012)
28. Ranise, S., Armando, A.: On the automated analysis of safety in usage control: A new decidability result. In: NSS. pp. 15–28 (2012)
29. Suter, P., Steiger, R., Kuncak, V.: Sets with cardinality constraints in satisfiability modulo theories. In: VMCAI. pp. 403–418 (2011)
30. Turkmen, F., den Hartog, J., Zannone, N.: Analyzing Access Control Policies with SMT. In: Proc. CCS. ACM (2014)
31. Zhang, N., Ryan, M., Guelev, D.P.: Synthesising verified access control systems through model checking. Journal of Computer Security 16(1), 1–61 (2008)
32. Zheng, Y., Zhang, X., Ganesh, V.: Z3-str: a Z3-based string solver for web application analysis. In: ESEC/SIGSOFT FSE. pp. 114–124 (2013)

8 Appendix A: Complexity of Policy Analysis with SMT

State-of-the-art SMT solvers combine a SAT solver with a *theory solver* for checking the satisfiability of *constraints*, i.e. conjunctions of atoms or their negations. The combination roughly works as follows: the atoms of the background theory are considered as Boolean variables so that the SAT solver can enumerate satisfying assignments for the formula being checked for satisfiability, mapping each propositional variable to either true or false. Each such assignment forms a constraint by taking the conjunction of any atom mapped to true and the negation of any atom mapped to false. If the theory solver confirms satisfiability of this constraint modulo the theory a solution has been found. Otherwise, the next assignment (if any) is attempted.

In the worst case, an exponential number of satisfying assignments must be considered. To address this theory solvers, in addition to checking satisfyability can be used to also derive new constraints which can be used to prune the search tree. The more complex work for the theory solver must be balanced against the gain from pruning and thus is usually required to maintain the same complexity as only checking satisfyability. This complexity depends on the theory; see, e.g., [5] for an overview. For instance, the theory of equality with uninterpreted functions (EUF) is decidable in polynomial time. Ground satisfiability in Linear Arithmetic (with variables ranging) over the Integers (LAI) is decidable and NP-complete. Note that LAI restricts the use of multiplication: variables can only be multiplied by a constant. Permitting multiplication of variables makes checking the satisfiability of the resulting constraints undecidable. In many practical situations, LAI constraints are of the form: $\pm x \pm y \leq c$ for c a constant. This belongs to the Unit-Two-Variable-Per-Inequality (UTVPI) fragment for which satisfiability is polynomial. Deriving all possible constraints in UTVPI, in particular those including disequalities, to prune the search space of a SAT solver is NP-complete. For this reason, theory solvers implemented in state-of-the-art SMT solvers are incomplete with respect to the capability of deriving implied constraints in UTVPI while checking for satisfiability. Other background theories that are often needed for the analysis of XACML policies (e.g., the theory of arrays, the theory of enumerated data types, and the theory of cardinality constraints on sets) are NP-complete. Despite the high theoretical complexity, efficient implementation of theory solvers for these theories exist which allow available SMT solvers to handle very large problem instances.

The situation is further complicated by the fact that more than one theory is needed to solve policy analysis problems for XACML policies. For instance, in the simple policy of Example 2, we have equality functions, numerical comparisons, operators over sets, etc. Under suitable assumption on the component theories, it is possible to build theory solvers capable of checking the satisfiability of constraints modulo the combination of theories by modularly re-using the theory solvers of the component theories; see again [5] for an overview. However, due to the need for the theory to interact, the complexity of the combination can be much higher than that of the individual theories. For example, if we combine

the theory of uninterpreted functions and UTVPI which both have polynomial complexity the result is NP-complete.

One may wonder if SMT tools are really needed to solve the logical problems resulting from XACML policy analysis problems. In fact, for many of the theories mentioned in Table 5, reductions to the Boolean satisfiability problem are available so that state-of-the-art SAT solvers would be sufficient. However, differentiating between SMT solvers and SAT solvers invoked after reductions does not seem meaningful from a conceptual point of view. Indeed, Kroening and Strichman [18] propose a framework in which the two approaches are instances, called eager and lazy, of the same reasoning strategy that consists in combining SAT solving with a reduction based on theory solving. A lazy strategy leads to interleaving SAT and theory solving along the lines discussed above. An eager strategy first applies theory reasoning in order to derive a Boolean formula which is equisatisfiable to the input quantifier-free formula and then invokes the SAT solver. It is also observed that eager and lazy strategies are on the same continuum and available implementations of state-of-the-art SMT solvers can freely mix the various approaches. However, to the best of our knowledge, almost all available tools are closer to implement the lazy strategy since this has emerged as the most efficient of the two. The upfront reduction to Boolean satisfiability of expressive theories (such as LAI) is computationally quite demanding since it requires encoding, as additional Boolean formulae, all implicit relations implied by the background theory. For instance, to enable a SAT solver detect the unsatisfiability of the formula $x = y \wedge y = z \wedge x \neq z$ modulo the theory of uninterpreted functions, it is necessary to add the formula $(x = y \wedge y = z) \Rightarrow x = z$ if $x = y$, $y = z$, and $x = z$ are seen as Boolean variables. Instead, the lazy strategy allows to perform reasoning in the theory on selected parts of the formula which are usually much smaller by interleaving Boolean and theory reasoning. Our experiments, discussed in Section 5, suggest that this is also the case for formulae generated from XACML policy analysis problems (see Table 1).

So far we have assumed that formulae to be checked for satisfiability modulo theories are quantifier-free. Yet example 2 show there can be constraints (e.g., ac_3) in XACML policies that use (universal) quantifiers. The decidability of quantifier-free formulae does not extend to quantified formulae. For instance, checking the satisfiability of quantified formulae modulo the theory of uninterpreted functions is undecidable since one can encode the satisfiability problem for arbitrary first-order formulae whose undecidability is well-known; see, e.g., [9]. Despite this and other negative results, a lot of efforts have been put in identifying classes of quantified formulae whose satisfiability is decidable by integrating instantiation or quantifier-elimination procedures in the combination of SMT and theory solving sketched above; see, e.g., [5] for pointers to relevant works. For instance, Presburger arithmetic extends LAI by allowing arbitrary quantification in formulae and is well-known to be decidable in double-exponential time [10] but recent work [7] has shown how to integrate a theory solver based on quantifier-elimination in a state-of-the-art SMT solver with very good performances on many problem instances. Unfortunately, when considering Presburger

arithmetic in combination with the theory of uninterpreted functions (as it is the case of our formulae), the satisfiability problem modulo the resulting theory is undecidable already in presence of a single unary predicate [14]. Fortunately, quantifier-instantiation procedures combined with the algorithm for checking the satisfiability of quantifier-free formulae (see, e.g., [12]) seem to work well in practice, i.e. for the kind of formulae resulting from several verification tasks. Our experience with a prototype implementation of the approach proposed in the paper confirms that this is also the case for the analysis of XACML policies.

We conclude this section by observing that certain policy analysis problems (such as those for attribute hiding and subsumption) require just one invocation of the SMT solver as it is sufficient to find one assignment of the attributes in the query to solve the problem (or proving that there is none). Other problems (such as those for change-impact and scenario-finding) instead require finding all assignments and thus require embedding the SMT solver in a loop to enumerate all models satisfying the formula. Indeed, the complexity for this kind of enumerative problems is higher. Fortunately, many available SMT tools offer support to implement the enumeration of models; see [5] for more on this issue.

9 Appendix B: Combining Algorithms

In this section, we provide the encodings of the most widely used combining algorithms of XACML, namely deny-overrides, permit-overrides and first-applicable. In particular, Table 3 presents the encoding of the combining algorithms as defined in XACML v3, and Table 4 presents the encoding of the combining algorithms as defined in XACML v2. In the tables, we use DS_{NA} to represent the *NotApplicable* space (i.e. $DS_{NA} = \overline{DS^p_D \cup DS^p_P \cup DS^p_{IN}}$).

Note that, differently from XACML v3, XACML v2 has two variants of deny-overrides and permit-overrides, one for combining rules and one for combining policies and policysets. Although all XACML v2 combining algorithms are defined over a four-valued decision set (i.e., *Permit*, *Deny*, *Indeterminate* and *NotApplicable*), deny-overrides and permit-overrides for rules implicitly record the effect of the rules that are evaluated *Indeterminate*. This is similar to the idea underlying the *Indeterminate* extended set used in XACML v3. In the encodings of Table 4, we use decision spaces $DS_{IN(D)}$ and $DS_{IN(P)}$ to make the effect of the rules evaluated *Indeterminate* explicit, where $DS_{IN(D)}$ indicates that the effect of the rule is *Deny* and $DS_{IN(P)}$ indicates that the effect of the rule is *Permit*. In deny-overrides combining algorithm for rules these decision spaces are defined as:
$DS^p_{IN(D)} = (DS^{p_1}_{IN(D)} \cup DS^{p_2}_{IN(D)}) \setminus (DS^p_D)$
$DS^p_{IN(P)} = (DS^{p_1}_{IN(P)} \cup DS^{p_2}_{IN(P)}) \setminus (DS^p_D \cup DS^p_{IN(D)} \cup DS^p_P)$,
while in permit-overrides combining algorithm for rules these decision spaces are defined as :
$DS^p_{IN(P)} = (DS^{p_1}_{IN(P)} \cup DS^{p_2}_{IN(P)}) \setminus DS^p_P$
$DS^p_{IN(D)} = (DS^{p_1}_{IN(D)} \cup DS^{p_2}_{IN(D)}) \setminus (DS^p_P \cup DS^p_{IN(P)} \cup DS^p_D)$.

Deny-overrides

$DS^p_D = DS^{p_1}_D \cup DS^{p_2}_D$

$DS^p_{IN(PD)} = \{(DS^{p_1}_{IN(PD)} \cup DS^{p_2}_{IN(PD)}) \cup [DS^{p_1}_{IN(D)} \cap (DS^{p_2}_{IN(P)} \cup DS^{p_2}_P)] \cup [DS^{p_2}_{IN(D)} \cap (DS^{p_1}_{IN(P)} \cup DS^{p_1}_P)]\} \setminus DS^p_D$

$DS^p_{IN(D)} = (DS^{p_1}_{IN(D)} \cup DS^{p_2}_{IN(D)}) \setminus (DS^p_D \cup DS^p_{IN(PD)})$

$DS^p_P = (DS^{p_1}_P \cup DS^{p_2}_P) \setminus (DS^p_D \cup DS^p_{IN(PD)} \cup DS^p_{IN(D)})$

$DS^p_{IN(P)} = (DS^{p_1}_{IN(P)} \cup DS^{p_2}_{IN(P)}) \setminus (DS^p_D \cup DS^p_{IN(PD)} \cup DS^p_{IN(D)} \cup DS^p_P)$

Permit-overrides

$DS^p_P = (DS^{p_1}_P \cup DS^{p_2}_P)$

$DS^p_{IN(PD)} = \{(DS^{p_1}_{IN(PD)} \cup DS^{p_2}_{IN(PD)}) \cup [DS^{p_1}_{IN(P)} \cap (DS^{p_2}_{IN(D)} \cup DS^{p_2}_D)] \cup [DS^{p_2}_{IN(P)} \cap (DS^{p_1}_{IN(D)} \cup DS^{p_1}_D)]\} \setminus DS^p_P$

$DS^p_{IN(P)} = (DS^{p_1}_{IN(P)} \cup DS^{p_2}_{IN(P)}) \setminus (DS^p_P \cup DS^p_{IN(PD)})$

$DS^p_D = (DS^{p_1}_D \cup DS^{p_2}_D) \setminus (DS^p_P \cup DS^p_{IN(PD)} \cup DS^p_{IN(P)})$

$DS^p_{IN(D)} = (DS^{p_1}_{IN(D)} \cup DS^{p_2}_{IN(D)}) \setminus (DS^p_P \cup DS^p_{IN(PD)} \cup DS^p_{IN(P)} \cup DS^p_D)$

First-applicable

$DS^p_D = DS^{p_1}_D \cup (DS^{p_1}_{NA} \cap DS^{p_2}_D)$

$DS^p_P = DS^{p_1}_P \cup (DS^{p_1}_{NA} \cap DS^{p_2}_P)$

$DS^p_{IN} = DS^{p_1}_{IN} \cup (DS^{p_1}_{NA} \cap DS^{p_2}_{IN})$

Table 3: Encoding of XACML v3 combining algorithms

Deny-overrides (Rules)

$DS^p_D = DS^{p_1}_D \cup DS^{p_2}_D$

$DS^p_P = (DS^{p_1}_P \cup DS^{p_2}_P) \setminus (DS^p_D \cup DS^p_{IN(D)})$

$DS^p_{IN} = (DS^p_{IN(D)} \cup DS^p_{IN(P)})$

Permit-overrides (Rules)

$DS^p_P = DS^{p_1}_P \cup DS^{p_2}_P$

$DS^p_D = (DS^{p_1}_D \cup DS^{p_2}_D) \setminus (DS^p_P \cup DS^p_{IN(P)})$

$DS^p_{IN} = (DS^p_{IN(P)} \cup DS^p_{IN(D)})$

First-applicable

$DS^p_D = DS^{p_1}_D \cup (DS^{p_1}_{NA} \cap DS^{p_2}_D)$

$DS^p_P = DS^{p_1}_P \cup (DS^{p_1}_{NA} \cap DS^{p_2}_P)$

$DS^p_{IN} = DS^{p_1}_{IN} \cup (DS^{p_1}_{NA} \cap DS^{p_2}_{IN})$

Deny-overrides (Policies)

$DS^p_D = DS^{p_1}_D \cup DS^{p_2}_D \cup DS^{p_1}_{IN} \cup DS^{p_2}_{IN}$

$DS^p_P = (DS^{p_1}_P \cup DS^{p_2}_P) \setminus DS^p_D$

Permit-overrides (Policies)

$DS^p_P = DS^{p_1}_P \cup DS^{p_2}_P$

$DS^p_D = DS^{p_1}_D \cup DS^{p_2}_D \setminus DS^p_P$

$DS^p_{IN} = (DS^{p_1}_{IN} \cup DS^{p_2}_{IN}) \setminus (DS^p_P \cup DS^p_D)$

Table 4: Encoding of XACML v2 combining algorithms

10 Appendix C: Mapping of Common XACML Functions to Background Theories

Table 5 presents a mapping of some of the common XACML functions to certain background theories that can encode the applicability constraints constructed from them. In the first and second columns, the class of functions and example instances from XACML specification are provided. The third column presents background theory/ies that can be used to model them while the fourth column shows the complexity of reasoning with the respective background theory/ies.

Table 5: XACML Functions and Background Theories

XACML Function Type	**Functions**	**Theory**	**Complexity**
Logical	*or*, *and*, *not*		
	n-of	Cardinality Constraints on Sets	NP-complete
Equality predicates	*integer-equal*, *double-equal*, *time-equal*, *date-equal*, *string-equal*	Enumerated Data Types **and/or** Equality with Uninterpreted Functions (EUF)	NP-complete
XPath-based	*xpath-node-count*	Equality with Uninterpreted Functions (EUF)	Polynomial
Regular-expression-based	*ipAddress-regexp-match*		
Arithmetic	*integer-add*, *integer-subtract*	Linear Arithmetic over the Integers (LAI)	NP-complete
Numeric comparison	*integer-greater-than*		
Date and time arithmetic	*dateTime-add-dayTimeDuration*		
Non-numeric comparison	(time) *time-in-range*, *time-less-than*		
	(string) *string-greater-than*	Theory of strings	NP-hard
String conversion	*string-normalize-space*		
Set	*type-intersection*, *type-union*	Theory of Arrays **and** Cardinality Constraints on Sets	NP-complete
Bag	*type-one-and-only*, *type-bag-size*		

Science Reports **Department of Mathematics and Computer Science**
Technische Universiteit Eindhoven

If you want to receive reports, send an email to: wsinsan@tue.nl (we cannot guarantee the availability of the requested reports).

In this series appeared (from 2012):

12/01	S. Cranen	Model checking the FlexRay startup phase
12/02	U. Khadim and P.J.L. Cuijpers	Appendix C / G of the paper: Repairing Time-Determinism in the Process Algebra for Hybrid Systems ACP
12/03	M.M.H.P. van den Heuvel, P.J.L. Cuijpers, J.J. Lukkien and N.W. Fisher	Revised budget allocations for fixed-priority-scheduled periodic resources
12/04	Ammar Osaiweran, Tom Fransen, Jan Friso Groote and Bart van Rijnsoever	Experience Report on Designing and Developing Control Components using Formal Methods
12/05	Sjoerd Cranen, Jeroen J.A. Keiren and Tim A.C. Willemse	A cure for stuttering parity games
12/06	A.P. van der Meer	CIF MSOS type system
12/07	Dirk Fahland and Robert Prüfer	Data and Abstraction for Scenario-Based Modeling with Petri Nets
12/08	Luc Engelen and Anton Wijs	Checking Property Preservation of Refining Transformations for Model-Driven Development
12/09	M.M.H.P. van den Heuvel, M. Behnam, R.J. Bril, J.J. Lukkien and T. Nolte	Opaque analysis for resource-sharing components in hierarchical real-time systems - extended version –
12/10	Milosh Stolikj, Pieter J. L. Cuijpers and Johan J. Lukkien	Efficient reprogramming of sensor networks using incremental updates and data compression
12/11	John Businge, Alexander Serebrenik and Mark van den Brand	Survival of Eclipse Third-party Plug-ins
12/12	Jeroen J.A. Keiren and Martijn D. Klabbers	Modelling and verifying IEEE Std 11073-20601 session setup using mCRL2
12/13	Ammar Osaiweran, Jan Friso Groote, Mathijs Schuts, Jozef Hooman and Bart van Rijnsoever	Evaluating the Effect of Formal Techniques in Industry
12/14	Ammar Osaiweran, Mathijs Schuts, and Jozef Hooman	Incorporating Formal Techniques into Industrial Practice
13/01	S. Cranen, M.W. Gazda, J.W. Wesselink and T.A.C. Willemse	Abstraction in Parameterised Boolean Equation Systems
13/02	Neda Noroozi, Mohammad Reza Mousavi and Tim A.C. Willemse	Decomposability in Formal Conformance Testing
13/03	D. Bera, K.M. van Hee and N. Sidorova	Discrete Timed Petri nets
13/04	A. Kota Gopalakrishna, T. Ozcelebi, A. Liotta and J.J. Lukkien	Relevance as a Metric for Evaluating Machine Learning Algorithms
13/05	T. Ozcelebi, A. Weffers-Albu and J.J. Lukkien	Proceedings of the 2012 Workshop on Ambient Intelligence Infrastructures (WAmIi)
13/06	Lotfi ben Othmane, Pelin Angin, Harold Weffers and Bharat Bhargava	Extending the Agile Development Process to Develop Acceptably Secure Software
13/07	R.H. Mak	Resource-aware Life Cycle Models for Service-oriented Applications managed by a Component Framework
13/08	Mark van den Brand and Jan Friso Groote	Software Engineering: Redundancy is Key
13/09	P.J.L. Cuijpers	Prefix Orders as a General Model of Dynamics

14/01	Jan Friso Groote, Remco van der Hofstad and Matthias Raffelsieper	On the Random Structure of Behavioural Transition Systems
14/02	Maurice H. ter Beek and Erik P. de Vink	Using mCRL2 for the analysis of software product lines
14/03	Frank Peeters, Ion Barosan, Tao Yue and Alexander Serebrenik	A Modeling Environment Supporting the Co-evolution of User Requirements and Design
14/04	Jan Friso Groote and Hans Zantema	A probabilistic analysis of the Game of the Goose
14/05	Hrishikesh Salunkhe, Orlando Moreira and Kees van Berkel	Buffer Allocation for Real-Time Streaming on a Multi-Processor without Back-Pressure
14/06	D. Bera, K.M. van Hee and H. Nijmeijer	Relationship between Simulink and Petri nets
14/07	Reinder J. Bril and Jinkyu Lee	CRTS 2014 - Proceedings of the 7th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems
14/08	Fatih Turkmen, Jerry den Hartog, Silvio Ranise and Nicola Zannone	Analysis of XACML Policies with SMT