sensors

MDPI

Article

A Formal Validation Approach for XACML 3.0 Access Control Policy

Carmine Caserio [1], Francesca Lonetti [2,*] and Eda Marchetti [2]

[1] Computer Science Department, University of Pisa, 56127 Pisa, Italy; carmcase2@gmail.com
[2] ISTI-CNR, 56124 Pisa, Italy; eda.marchetti@isti.cnr.it
* Correspondence: francesca.lonetti@isti.cnr.it

Abstract: Access control systems represent a security mechanism to regulate the access to system resources, and XACML is the standard language for specifying, storing and deploying access control policies. The verbosity and complexity of XACML syntax as well as the natural language semantics provided by the standard make the verification and testing of these policies difficult and error-prone. In the literature, analysis techniques and access control languages formalizations are provided for verifiability and testability purposes. This paper provides three contributions: it provides a comprehensive formal specification of XACML 3.0 policy elements; it leverages the existing policy coverage criteria to be suitable for XACML 3.0; and it introduces a new set of coverage criteria to better focus the testing activities on the peculiarities of XACML 3.0. The application of the proposed coverage criteria to a policy example is described, and hints for future research directions are discussed.

Keywords: XACML 3.0 formalization; coverage criteria; policy testing

Citation: Caserio, C.; Lonetti, F.; Marchetti, E. A Formal Validation Approach for XACML 3.0 Access Control Policy. *Sensors* **2022**, *22*, 2984. https://doi.org/10.3390/s22082984

Academic Editor: Nikos Fotiou

Received: 28 March 2022
Accepted: 9 April 2022
Published: 13 April 2022

1. Introduction

Security is a challenging issue in modern networked systems where a huge amount of data are managed and exchanged in everyday life. In particular, the privacy and confidentiality attributes of personal and critical data require adequate security mechanisms to be put in place [1]. In this context, access control systems represent an important component for the overall security because they are able to mediate all requests of access. This ensures the protection of data and assures that only the intended, i.e., authorized users can access them. In particular, specific rules can be defined for establishing under which conditions a subject's access request to a resource can be permitted or denied.

In this context, attribute-based access control (ABAC) [2] systems are the adopted means for enhancing a fine-grained access control. ABAC relies on the combination of various attributes of authorization elements into access control decisions. In the literature, there are several languages for specifying access control policies [3], and among them, the OASIS eXtensible Access Control Markup Language (XACML) [4] is the most commonly used standard in many real-world systems for defining ABAC policies in the XML-based syntax. The XACML language is also widely used to guarantee access control decisions for the distributed Internet of Things (IoT) environments [5,6].

The management of real access control policies is in practice difficult and error-prone [7] due to the verbosity and complexity of the XACML syntax. Faults in the access control policies are very critical because they could open the path to security flaws: either denying accesses that should be authorized or allowing accesses to non-authorized users.

Thus, verification and validation become key issues for XACML policies specification and their implementation [8].

However, in the literature, the commonly available test cases' generation approaches for XACML policies leverage combinatorial methodologies [9–12]. With the adoption of

these methodologies, the generated number of test cases rapidly grows according to the policy complexity.

The execution of a large number of test cases can drastically increase the cost and effort of the testing phase especially for what concerns the oracle definition, i.e., checking the test results and deciding whether they are correct or not.

In practice, in the context of access control systems, except for some attempts to automatically derive the oracle from the XACML policy model [13], this oracle derivation is usually performed manually, because the complexity of the XACML language makes the use of automated support very difficult. Given the real constraints on testing budget, a key issue in the testing of XACML policies is to reduce as much as possible the number of tests to be executed while trying to maximize the fault detection effectiveness of the reduced test suite.

The main goal of this paper is to address this specific issue by proposing some testing coverage criteria specifically conceived on the peculiarities of the XACML 3.0 language that is the current standard language for access control policies definition. To define these XACML 3.0-based coverage criteria, we first provide a formal definition of the main elements of the XACML 3.0 language. The proposed XACML 3.0-based coverage criteria will guide the testing activities and enhance the trustworthiness of XACML-based access control systems.

As stated in the literature, coverage information can provide an indication of the effectiveness of the executed test cases and can guide the generation of test cases [14]. However, some empirical results [15] show that the performance of the reduced test suite could vary according to the considered program and the adopted coverage criterion.

In this paper, we provide some examples of how the proposed coverage criteria could be useful in addressing problems and inconsistencies in policy specification. In particular, we start from a formal definition of XACML syntax and semantics, and then, we show its use for enabling the understanding of XACML standard intentions and allow rigorous analysis techniques for verifiability and testability purposes.

Some attempts to provide formal reasoning techniques for the analysis and verification of policies have been provided [16–20]. They mainly focus on specific aspects of the language or design new expressive languages whose formal foundations enable tool-supported analysis and enforcement of access control policies.

In this paper, starting from the work of [16], we refine the definition of the abstract syntax and semantics of XACML 3.0 standard. According to this formal definition, we define a formal specification of the coverage process of the elements of the XACML policy and provide some coverage criteria that is useful to assess the XACML policy.

In particular, we select as the specification language the abstract syntax of the Context Free Grammar (CFG) [21], because it provides a precise mathematical definition that clearly rules out the XACML language. Additionally, the formalization provided by the CFG is a machine readable specification, which can be easily implemented in an automatic way.

Summarizing, the main contributions of this paper are: (i) to derive a rigorous formalization of XACML 3.0 standard leveraging context-free grammar [21], extending and revising the proposal of [16], and (ii) to formally define some policy coverage criteria for XACML 3.0 policy testing.

We also discuss possible ways to adopt the defined coverage criteria for improving the verifiability and testability of the policy such as: (i) the derivation of XACML requests according to the defined coverage criteria; (ii) the reduction (or selection) of a given test suite according to defined coverage criteria and its effect on fault detection; (iii) finally, the coverage measurement of the XACML policy in order to detect uncovered or redundant parts of the policy as well as guide the development of further test cases, which can improve the quality of policy testing.

The remainder of this paper is organized as follows. Section 2 introduces the XACML language. Section 3 presents the related work. Section 4 presents the formalization of XACML 3.0 policy elements, while Section 5 presents the formalization of the XACML

request. Section 6 presents the coverage definition of XACML 3.0 policy elements, whereas Section 7 shows the definition of some coverage criteria and their application to a policy example. Finally, Section 8 concludes the paper, also hinting at future work.

2. Access Control Policies in XACML 3.0

XACML 3.0 language is a de facto standardized specification language for ruling the system access in an XML format. It relies on two main concepts: the XACML policy used for modeling the access requirements of a protected system; and the XACML request used for requiring the access to a protected resource. In the access control system, the request is evaluated against the policy to allow (or deny) the access to the resource. In the following, more details about these two concepts are provided.

An XACML policy is characterized by a tree structure having as its root the *PolicySet* and as children one (or more) *PolicySet*(s) or *Policy* elements. The latter includes: a *Target*, which specifies the execution constraints in terms of the subjects, resources, actions, and environments on which the policy can be applied; and a set of rules having also in turn a target, a condition and a rule-combining algorithm. Usually, the target is represented by a conjunctive sequence of *AnyOf* clauses. In particular, each *AnyOf* clause is a disjunctive sequence of *AllOf* clauses, and each *AllOf* clause is a conjunctive sequence of match predicates, and each match predicate compares attribute values in a request with the policy attributes. Both match predicates and rule conditions use different logical expressions and a variety of predefined functions and data types on subject, resource, action, and environment.

Only when a request satisfies the target of the *PolicySet* or *Policy*, the associated set of rules can be evaluated; otherwise, it is skipped.

Considering in detail the structure of a rule, its main elements are: the *Target* and the *Condition*, i.e., a set of Boolean functions used for establishing when the request is applicable to the rule. In this last case, the outcome of the rule is the rule effect (*Permit* or *Deny*). If the request is not applicable to the rule, the evaluation outcome is *NotApplicable* or *Indeterminate* in case of error.

The combining algorithms (either the *PolicyCombiningAlgorithm* or *RuleCombiningAlgorithm*) define how to combine the evaluation results in order to provide a unique evaluation outcome (access result). As an example, the *deny-overrides algorithm* establishes that *Deny* takes the precedence regardless of any other rules evaluation. Therefore, it will return *Deny* if there is a rule that is evaluated to *Deny*. It will return *Permit* if there is at least a rule that is evaluated to *Permit*, and all the other rules are evaluated to *NotApplicable*. Alternatively, an *Indeterminate* result is provided as the outcome if there is an error in the evaluation of a rule with the *Deny* effect, and the other policy rules with the *Deny* effect are not applicable.

Similarly, the *first-applicable* algorithm forces the rules evaluation in the order in which they are listed in the policy. The final outcome will be the effect of the first applicable rule (i.e., *Permit* or *Deny*). In all the cases, if the evaluation of the rule target (or the rule condition) is *False*, the next rule in the order will be evaluated until no further rule in the order exists, and the final *NotApplicable* result is provided.

For the evaluation, all attribute and element values describing the subject, resource, action, and environment of an access request are considered and compared with the attribute and element values of the policy.

The above-described access control mechanism relies on the standardized access control system architecture represented in Figure 1. As in the Figure, the main entities are:

- the Policy Enforcement Point (PEP) that is in charge of receiving the user's request, transforming it into an XACML request and sending it to the PDP. It also allows or denies the access to the resource.
- the Policy Decision Point (PDP) which is in charge of the request evaluation against the policy and the computation of the access response (Permit/Deny/NotApplicable). The PDP retrieves the access policies from the Policy Administration Point (PAP) and the attributes values from the Policy Information Point (PIP).

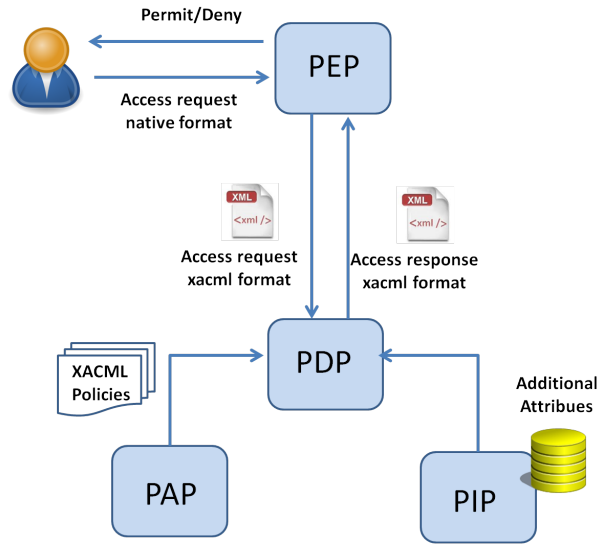


Figure 1. XACML Basic Model.

3. Related Work

This work spans over two main research directions: XACML formalization (Section 3.1) and coverage criteria for testing purposes (Section 3.2). Moreover, in Section 3.3, the specific advancements of our work with respect to the SotA are presented.

3.1. XACML Formalization

Access control languages formalization has recently been the subject of extensive research, and several attempts have been provided to analyze and formalize XACML policies.

Margrave is a popular propositional logic analysis tool for XACML policy verification [22]. Logic programming systems and description logics (DL) have been used to model XACML policy constraints and roles hierarchies [23,24]. Specifically, the work in [24] extends the analysis services offered by Margrave leveraging full First-Order logic XACML analysis tools (such as Alloy [25]). It is able to cover the analysis of role hierarchies and role cardinality as well as analysis of policy redundancy.

Bryans explores the use of process algebra for formalizing and analyzing XACML language, presenting the core concepts of XACML using Communicating Sequential Processes (CSP) [26]. Other woks propose different formalization approaches for capturing the semantics of XACML constructs [27–29] and specifying an access control meta-model to derive access control policies or rule-based policies [30–32].

Other approaches to enable the automated verification of access control policies rely on SAT solver. Specifically, the authors of [33] present a formal model defining the semantics of the XACML access control language; then, they define ordering relations on access control policies that are used to automatically verify properties of the policies by translating them to boolean satisfiability problems and then applying a sat solver.

Masi et al. in [17] define a formal expressive access control policy language denoted as FACPL, supporting automated specification, analysis, and the enforcement of access control policies. It refines and extends XACML language and relies on a rigorously defined denotational semantics that allows for the management of missing attributes and formalization of combining algorithms. For enabling automated policies analysis, they introduce a constraint formalism based on Satisfiability Modulo Theories (SMT) formulae, which has been proven to be more effective than other ones, such as decision diagrams or description logic.

A more recent work [13] aims to formalize XACML policies by using a typed graph, called the XAC-Graph. Then, this XAC-Graph is used for both XACML requests generation and automated model-based oracle derivation.

Finally, the work in [16] differently from the previous approaches focuses on the last standard version of XACML that is XACML 3.0 and provides a formally defined semantics for XACML 3.0 components evaluation and standard combining operators, proposing new directions for the XACML standard extension.

Our proposal revises and extends the work of [16], proving a complete definition of the abstract syntax and semantics of XACML 3.0 standard, leveraging context-free grammars. Moreover, differently from all the mentioned formalization approaches, our work focuses more on policy testing than the verification of policies properties. For this, we use the proposed formalization to define some policy coverage criteria that can enhance the effectiveness of policy testing activity.

3.2. Coverage-Based Policy Testing

Coverage metrics represent an effective way to assess the test quality and compare different testing solutions. The aim of adequacy criteria is to evaluate a specific testing strategy by measuring the percentage of the exercised elements in the program or in its specification.

In the literature, test coverage is adopted for different purposes: (i) enhancing the test suite with additional test cases in order to exercise elements that have not been tested; (ii) test set augmentation and test set minimization in the context of regression testing [34]; (iii) selection of test cases and evaluation of test cases effectiveness [35]; (iv) test cases prioritization [36], aiming to reorder test cases so that the tests with a higher priority can be executed before those with a lower priority; and finally, (v) software maintenance [34]. The authors of [34] propose a systematic survey of coverage-based testing, whereas the work in [37] focuses on coverage criteria for state-based testing.

Many solutions for test coverage measurement and analysis have been proposed. They are based on the adopted policy specification language. The work in [38] provides a first attempt of coverage criterion for XACML policies. It defines three different structural coverage metrics able to reduce the generated test sets and validates the effects of this test's reduction in terms of fault detection.

An extension of the coverage solution presented in [38] is proposed in [39]. This work defines an XACML-based smart coverage selection approach that focuses on the policy rules coverage and provides a formalization of the proposed coverage criterion relying on the *Rule Target Set* concept, i.e., the union of the target of the rule, and all enclosing policy and policy sets targets. The main concept of the proposed criterion is that in order to match the rule target, the requests must first match the enclosing policy and policy sets targets.

Cirg (Change-Impact Request Generation) [40] is for instance a framework able to generate access requests through the change-impact analysis of two synthesized versions of an XACML policy and allows a reduction of the number of tests based on policy structural coverage. The work in [41] leverages mutation analysis and coverage analysis to perform regression testing of security policies.

More recently, in [12], a family of coverage criteria for XACML policies, including Modified Condition/Decision Coverage (MC/DC), has been presented and evaluated through mutation analysis in order to establish the most effective one in terms of fault detection. In [42], a proposal for the continuous tracing of policy execution and corresponding coverage criteria has been presented in order to detect inconsistencies in the policy specification and provide support for policies updates if new events occur.

Differently from previous solutions, our proposal provides a rigorous formalization of coverage of XACML 3.0 elements and formally specifies four additional coverage criteria specifically conceived for XACML 3.0 policy testing purposes.

3.3. SotA Advancements

In order to clarify the position of our paper with respect to the State of the Art, we report the analysis of the related works in the last 10 years in Table 1. In particular, in the first column, we provide the reference of the paper and the publication year; in the column

labeled *Paper contribution*, we summarize the main contribution of the considered related work; in the column labeled *Language/Formalism*, we report the target specification language; in the columns labeled *Access control formalization* and *Coverage metrics/ measures*, we specify if the paper focuses on the formalization or on the coverage metrics and measures, respectively. Finally, in the last column labeled *Our advancement with respect to SotA*, we provide the advancements of our paper with respect to the considered related work. In the last row of the table, we provide the classification of our contribution.

As shown in the table, most of the related works focus on the possible formalization of the access control policy and its constructs either considering XACML or ABAC/R-BAC specification language. Coverage criteria and metrics have been rarely analyzed or improved during these last 10 years.

As evidenced by the table, the related works are strictly divided into two categories: either they focus on the formalization or they provide coverage criteria. To the best of our knowledge, as evidenced in the last row of the table, our paper is the first attempt to combine the two categories with the purpose of providing a comprehensive process useful for both developers and testers.

Table 1. SotA Advancements.

Paper (Year)	Paper Contribution	Language/ Formalism	Access Control Formalization	Coverage Metrics/ Measures	Our Advancement with Respect to SotA
[17] (2012)	The paper provides a formal semantics of XACML and its implementation based on such semantics	XACML 2.0	✓		New formalization of the XACML 3.0 standard language by considering context-free grammars as an alternative representation
[39] (2014)	The paper provides a test selection approach based on a coverage measure	XACML 2.0		✓	New coverage criteria provided and revision of the existing ones to XACML 3.0
[16] (2014)	The paper provides a formalization of the XACML3.0 using Belnap logic and D-algebra	XACML 3.0	✓		New formalization of the XACML 3.0 standard language by considering context-free grammars as an alternative representation
[27] (2015)	The paper provides a formalization of semantic differences between the combining algorithms	XACML 3.0	✓		Formalization extended to the overall structure of XACML
[28] (2015)	The paper proposes a UML profile for XACML policies specification	XACML 3.0 UML	✓		New formalization of the XACML 3.0 standard language by considering context-free grammars as an alternative representation
[42] (2018)	The paper proposes an access control infrastructure, based on a monitor engine enabling coverage criterion selection	XACML 2.0		✓	New conceived coverage criteria that could be used for improving the monitor engine. This is part of our future work
[12] (2018)	The paper defines a family of coverage criteria for XACML policies, including the MC/DC criterion	XACML 3.0		✓	Formalization of the coverage definitions considering the context-free grammars and focusing on the specific XACML 3.0 elements. Formalization of MC/DC criterion could be part of future work

Table 1. *Cont.*

Paper (Year)	Paper Contribution	Language/ Formalism	Access Control Formalization	Coverage Metrics/ Measures	Our Advancement with Respect to SotA
[43] (2018)	The paper proposes a direct logical formalism of ABAC models using a variant of description logics and function-free first-order logic with equality	ABAC RBAC	✓		New formalization of the XACML 3.0 standard language by considering context-free grammars as an alternative representation
[29] (2020)	The paper formally models the resource attributes by dynamic description logic (DDL) and then provides means for rules conflict solving	XACML 3.0	✓		New formalization of the XACML 3.0 standard language by considering context-free grammars as an alternative representation
[31] (2021)	The paper proposes a generic AC metamodel for defining AC policies	ABAC RBAC	✓		New formalization of the XACML 3.0 standard language by considering context-free grammars as an alternative representation
[30] (2021)	The paper provides a theory for representing the semantics of rule-based policies based on the semantics of conditional expressions in their rules	ABAC RBAC	✓		New formalization of the XACML 3.0 standard language by considering context-free grammars as an alternative representation
[32] (2022)	The paper proposes an access control metamodel useful to derive various instances of AC models	ABAC RBAC	✓		New formalization of the XACML 3.0 standard language by considering context-free grammars as an alternative representation
This paper (2022)	The paper provides (i) a formalization of the XACML 3.0 standard language leveraging context-free grammars (ii) the formal definition of the coverage of the XACML 3.0 elements (iii) the definition of several coverage criteria specifically conceived for XACML 3.0 policies for testing purposes	ABAC RBAC	✓	✓	-

4. Formalization of Primary XACML 3.0 Elements

In this section, we provide the formalization of the XACML standard for describing security access control policies. Starting from the seminal work of [16], we provide here the definition of the abstract syntax of XACML 3.0 standard (Section 4.1), the specification of the XACML grammar (Section 4.2), and an example of policy formalization (Section 4.3).

4.1. XACML 3.0 Primary Elements

In this section, we introduce the alphabet of the grammar we use to describe the structure of an XACML access control policy. In particular, we extend and revise the XACML syntax introduced in [16], so to better represent the coverage concepts of this paper. For aim of completeness, we report in Table 2 the symbols we use in this paper for the definition of the grammar.

Table 2. Symbols of the grammar.

PS	`PolicySet` element
T_{PS}	`Target` of the `PolicySet` element
P	`Policy` element
PCA	`PolicyCombiningAlgorithm`
T_P	`Target` of the `Policy` element
A_{nyOf}	`AnyOf` element
A_{llOf}	`AllOf` element
M	`Match` element
$A_{ttrChoice}$	Syntactic category used for deciding between a_d and a_s
A_{ttr}	`Attribute`
S_{ubject}	`Subject`'s attribute
$R_{esources}$	`Resource`'s attribute
A_{ction}	`Action`'s attribute
E_{nv}	`Environment`'s attribute
A_{dHoc}	`User`'s attribute
a_v	`AttributeValue`
a_d	`AttributeDesignator`
a_s	`AttributeSelector`
s	`Subject`'s instance
r	`Resource`'s instance
a	`Action`'s instance
e	`Environment`'s instance
a_h	Ad hoc `User`'s attribute instance
R	`Rule` element
T_R	`Target` of the `Rule` element
RCA	`RuleCombiningAlgorithm`
C	`Condition` element
E_{Xbool}	Syntactic category $\subset E_X$ category for producing only expressions that obtain a boolean result
A	`Apply` element
E_X	`Expression` element
E_{Xfun}	Syntactic category $\subset E_X$ category for producing only expressions that are functions
F_{un}	`Function` element
E_{ffect}	`Effect` that can be p_{ermit} or d_{eny}

An initial formalization of the syntactical categories is the one illustrated in Table 3. It focuses on the syntactical sets that can be identified in an XACML access control specification and uses the Kleene star operator for deciding whether a string belongs to a grammar or not.

Table 3. Definition of syntactical categories with formalization on sets.

$\mathcal{PS}$	$=$	$\{\mathcal{P}\}^+ \cup \{\mathcal{PCA}\} \cup \{\mathcal{T}_{\mathcal{PS}}\}$
P	$=$	$\{\mathcal{T}_{\mathcal{P}}\} \cup \{\mathcal{R}\}^+ \cup \{\mathcal{RCA}\}$
$\mathcal{T}_{\{\mathcal{PS},\,\mathcal{P},\,\mathcal{R}\}}$	$=$	$\{\text{AnyOf}\}^*$
AnyOf	$=$	$\{\text{AllOf}\}^+$
AllOf	$=$	$\{\mathcal{M}\}^+$
$\mathcal{M}$	$=$	$\left\{\{a_v\} \cup \{el : el \in \{a_d \cup a_s\}\}\right\}^+$
$\mathcal{R}$	$=$	$\{E_{ffect}\} \cup \{\mathcal{T}_{\mathcal{R}}\} \cup \{\mathcal{C}\}$
$E_{\mathcal{X}}$	$=$	$\{\mathcal{A}, a_v, a_s, a_d, \mathcal{F}_{un}\}$
$\mathcal{C}$	$=$	$\{\exists!\ e \in E_X : e_{xpr}val(e) = \begin{cases} T_{rue} \\ F_{alse} \end{cases}\}$
$\mathcal{A}$	$=$	$\{\mathcal{F}_{un}\} \cup \{E_{\mathcal{X}}\}^*$
$\mathcal{F}_{un}$	$=$	$\{\mathcal{F}_{un_{id}}\}$

According to this formalization, given a policy $\mathcal{P}$ having a set of n rules $\mathbf{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$, and a Policy target $\mathcal{T}_\mathcal{P}$ possibly empty, the Boolean satisfiability (SAT) of the policy can be defined as:

$$\begin{cases} (\mathcal{P} = \{\mathbf{R}\}) \Rightarrow SAT(\mathbf{R}) & \text{if } \mathcal{T}_\mathcal{P} = \epsilon \\ (\mathcal{P} = \{\mathcal{T}_\mathcal{P}, \mathbf{R}\}) \Rightarrow (SAT(\mathcal{T}_\mathcal{P}) \Rightarrow SAT(\mathbf{R})) & \text{otherwise} \end{cases}$$

In the similar way, the Boolean satisfiability (SAT) of each of the policy rules, generically indicated with $\mathcal{R}$, can be defined as:

$$\begin{cases} (\mathcal{R} = \emptyset) \Rightarrow t_{rue} \\ \text{if } \mathcal{T}_\mathcal{R} = \epsilon \wedge \mathcal{C} = \epsilon \\ (\mathcal{R} = \{\mathcal{C}\}) \Rightarrow (SAT(\mathcal{T}_\mathcal{P}) \Rightarrow eval(\mathcal{C})) \\ \text{if } \mathcal{T}_\mathcal{R} = \epsilon \\ (\mathcal{R} = \{\mathcal{T}_\mathcal{R}\}) \Rightarrow SAT(\mathcal{T}_\mathcal{R}) \\ \text{if } \mathcal{C} = \epsilon \\ (\mathcal{R} = \{\mathcal{T}_\mathcal{R}\} \cup \{\mathcal{C}\}) \Rightarrow (SAT(\mathcal{T}_\mathcal{R}) \Rightarrow eval(\mathcal{C})) & \text{otherwise} \end{cases}$$

4.2. XACML 3.0 Grammar

Formally, a context-free grammar [21] is defined by a quadruple:

$$G = \langle \Lambda, V, S, P \rangle$$

where

- Λ is the set of symbols, called alphabet;
- V is the finite set of the syntactical categories;
- S is the main syntactical category;
- P is the set of all productions, where each production, in the case of context-free grammars (which is the one we need for `XACML 3.0`) is a finite relation:

$$A \rightarrow \alpha$$

where $A \in V$, $\alpha \in (\Lambda \cup V)^*$ (the asterisk represents the Kleene star).

Applying the above definition to the `XACML 3.0` access control language, we have: the set Λ is a subset of all the possible elements of the `XACML 3.0`, and it is defined as

$$\Lambda = V_0 \cup \{a_v, a_s, a_d, s, r, a, e, a_h, p_{ermit}, d_{eny}, t_{rue}, f_{alse},$$
$$p_{ermit}o_{verrides}, d_{eny}o_{verrides}, f_{irst}a_{pplicable},$$
$$o_{nly}o_{ne}a_{pplicable}, d_{eny}u_{nless}p_{ermit}, p_{ermit}u_{nless}d_{eny},$$
$$o_{rdered}d_{eny}o_{verrides}, o_{rdered}p_{ermit}o_{verrides}\}$$

where

$$V_0 = \{P_S, T_{PS}, P, P_{CA}, T_P, A_{nyOf}, A_{llOf}, M, A_{ttrChoice},$$
$$A_{ttr}, R, R_{CA}, T_R, C, E_{Xbool}, A, E_X, E_{Xfun}, F_{un},$$
$$E_{ffect}, S_{ubject}, R_{esource}, A_{ction}, E_{nv}, A_{dHoc}\}$$

The main syntactical category S is defined as:

$$S = \{P_S\}$$

The set of all productions P is defined in Table 4.

Table 4. Productions of `XACML 3.0` grammar.

$$
\begin{array}{lcl}
P = \{ & & \\
 & P_S \rightarrow & P_S\ P \mid P_{S_1}\ P_S \mid T_{PS}\ P_{CA}\ P \\
 & T_{PS} \rightarrow & A_{nyOf}\ T_{PS} \mid \epsilon \\
 & P \rightarrow & R_{CA}\ T_P\ R \mid P\ R \\
 & P_{CA} \rightarrow & p_{ermit}o_{verrides} \mid d_{eny}o_{verrides} \mid \\
 & & f_{irst}a_{pplicable} \mid \\
 & & o_{nly}o_{ne}a_{pplicable} \mid \\
 & & d_{eny}u_{nless}p_{ermit} \mid \\
 & & p_{ermit}u_{nless}d_{eny} \mid \\
 & & o_{rdered}d_{eny}o_{verrides} \mid \\
 & & o_{rdered}p_{ermit}o_{verrides} \\
 & T_P \rightarrow & A_{nyOf}\ T_P \mid \epsilon \\
 & A_{nyOf} \rightarrow & A_{llOf}\ A_{nyOf} \mid A_{llOf} \\
 & A_{llOf} \rightarrow & M\ A_{llOf} \mid M \\
 & M \rightarrow & F_{un}\ a_v\ A_{ttrChoice}\ M \mid \\
 & & F_{un}\ a_v\ A_{ttrChoice} \\
 & A_{ttrChoice} \rightarrow & a_s \mid a_d \\
 & R_{CA} \rightarrow & p_{ermit}o_{verrides} \mid \\
 & & d_{eny}o_{verrides} \mid \\
 & & f_{irst}a_{pplicable} \mid \\
 & & d_{eny}u_{nless}p_{ermit} \mid \\
 & & p_{ermit}u_{nless}d_{eny} \mid \\
 & & o_{rdered}d_{eny}o_{verrides} \mid \\
 & & o_{rdered}p_{ermit}o_{verrides} \\
 & R \rightarrow & E_{ffect}\ T_R\ C \\
 & T_R \rightarrow & T_R\ A_{nyOf} \mid \epsilon \\
 & E_X \rightarrow & A \mid a_s \mid a_v \mid a_d \mid F_{un} \\
 & E_{Xfun} \rightarrow & F_{un} \\
 & F_{un} \rightarrow & s_{tring}e_{qual} \mid s_{tring}g_{reater}t_{han} \mid \\
 & & s_{tring}l_{ess}t_{han} \mid \\
 & & s_{tring}g_{reater}t_{han}o_re_{qual} \mid \\
 & & s_{tring}l_{ess}t_{han}o_re_{qual} \\
 & & i_{nteger}e_{qual} \mid i_{nteger}g_{reater}t_{han} \mid \\
 & & i_{nteger}l_{ess}t_{han} \mid \\
 & & i_{nteger}g_{reater}t_{han}o_re_{qual} \mid \\
 & & i_{integer}l_{ess}t_{han}o_re_{qual} \\
 & E_{ffect} \rightarrow & p_{ermit} \mid d_{eny} \\
\} & &
\end{array}
$$

4.3. Example of Policy and Its Formalization

In this section, we provide an example of policy representation through the grammar introduced in the previous section. The sample policy considered is shown in Listing 1. In this case, starting from $S = \{\mathcal{PS}\}$, the policy can be expressed by the following production:

1–160 In Listing 1, the `PolicySet` element contains: two `Policy` elements (`Policy1` and `Policy2` at line 30 and line 96 of Listing 1 respectively), the declaration of the policy-combining algorithm considered that in this case is first applicable (line 8), and a policy set `Target` element (lines 12–29 of Listing 1) that specifies that the subject element must be an integer less than 15. This can be expressed as:

$$\boxed{\boxed{PS}} \rightarrow \boxed{PS}\ P_2 \rightarrow \boxed{T_{PS}}\ \boxed{P_{CA}}\ P_1\ P_2 \rightarrow \boxed{A_{nyOf}}\ F_{irst}A_{pplicable}\ P_1\ P_2 \rightarrow$$

$$\boxed{A_{llOf}}\ F_{irst}A_{pplicable}\ P_1\ P_2 \rightarrow$$

$$\boxed{M}\ F_{irst}A_{pplicable}\ P_1\ P_2 \rightarrow \boxed{F_{un}}\ a_v\ \boxed{A_{ttr}C_{hoice}}\ F_{irst}A_{pplicable}\ P_1\ P_2 \rightarrow$$

$$\boxed{i_{nteger}l_{ess}t_{han}\ a_v\ a_s\ F_{irst}A_{pplicable}\ \boxed{P_1}\ \boxed{P_2}}$$

30–95 Each of the policies in the policy set has in turn a `Target` element and a `Rule` element. In particular, `Policy1` specifies that the rule-combining algorithm considered is first applicable (line 32), the policy `Target` element (lines 39–56 of listing 1) specifies that the subject element must be equal to the integer `10`. This can be expressed as:

$$\boxed{\boxed{P_1}} \rightarrow \boxed{R_{CA}} \ \boxed{T_{P_1}} \ R_1 \rightarrow F_{irst}A_{pplicable} \ \boxed{A_{nyOf}} \ R_1 \rightarrow F_{irst}A_{pplicable} \ \boxed{A_{llOf}}$$
$$R_1 \rightarrow F_{irst}A_{pplicable} \ \boxed{M} \ R_1 \rightarrow F_{irst}A_{pplicable} \ \boxed{F_{un}} \ a_v \ \boxed{A_{ttr}C_{hoice}} \ R_1 \rightarrow$$

$$\boxed{\boxed{F_{irst}A_{pplicable} \ i_{nteger}e_{qual} \ a_v \ a_s \ \boxed{R_1}}}$$

57–94 The `Rule1` element returns the `Deny` effect (line 58) in case: (i) the subject element is greater than `4` as declared in the rule `Target` (lines 59–77 of Listing 1) and (ii) the subject element is greater than `2` as declared in the rule `Condition` (lines 78–93 of Listing 1). This can be expressed as:

$$\boxed{\boxed{R_1}} \rightarrow \boxed{E_{ffect}} \ \boxed{T} \ C \rightarrow D_{eny} \ \boxed{A_{nyOf}} \ C \rightarrow D_{eny} \ \boxed{A_{llOf}} \ C \rightarrow D_{eny} \ \boxed{M} \ C \rightarrow$$
$$D_{eny} \ \boxed{F_{un}} \ a_v \ \boxed{A_{ttr}C_{hoice}} \ C \rightarrow D_{eny} \ i_{nteger}g_{reater}t_{han} \ a_v \ a_d \ \boxed{C} \rightarrow$$
$$D_{eny} \ i_{nteger}g_{reater}t_{han} \ a_v \ a_d \ \boxed{E_{X_{bool}}} \rightarrow D_{eny} \ i_{nteger}g_{reater}t_{han} \ a_v \ a_d \ \boxed{A} \rightarrow$$
$$D_{eny} \ i_{nteger}g_{reater}t_{han} \ a_v \ a_d \ \boxed{A} \ \boxed{E_X} \rightarrow D_{eny} \ i_{nteger}g_{reater}t_{han} \ a_v \ a_d \ \boxed{A} \ \boxed{E_X} \ a_d \rightarrow$$
$$D_{eny} \ i_{nteger}g_{reater}t_{han} \ a_v \ a_d \ \boxed{E_{X_{fun}}} \ a_v \ a_d \rightarrow$$

$$\boxed{D_{eny} \ i_{nteger}g_{reater}t_{han} \ a_v \ a_d \ i_{nteger}g_{reater}t_{han} \ a_v \ a_d}$$

96–159 Similarly, `Policy2` specifies that the rule-combining algorithm considered is first applicable (line 97), whereas the `Target` element (lines 104–121 of Listing 1) specifies that the subject element must be less than the integer `9` and there is a `Rule` element. This can be expressed as:

$$\boxed{\boxed{P_2}} \rightarrow \boxed{R_{CA}} \ \boxed{T_{P_2}} \ R_2 \rightarrow F_{irst}A_{pplicable} \ \boxed{A_{nyOf}} \ R_2 \rightarrow F_{irst}A_{pplicable} \ \boxed{A_{llOf}} \ R_2 \rightarrow$$
$$F_{irst}A_{pplicable} \ \boxed{M} \ R_2 \rightarrow F_{irst}A_{pplicable} \ \boxed{F_{un}} \ a_v \ \boxed{A_{ttr}C_{hoice}} \ R_2 \rightarrow$$

$$\boxed{F_{irst}A_{pplicable} \ i_{nteger}l_{ess}t_{han} \ a_v \ a_s \ \boxed{R_2}}$$

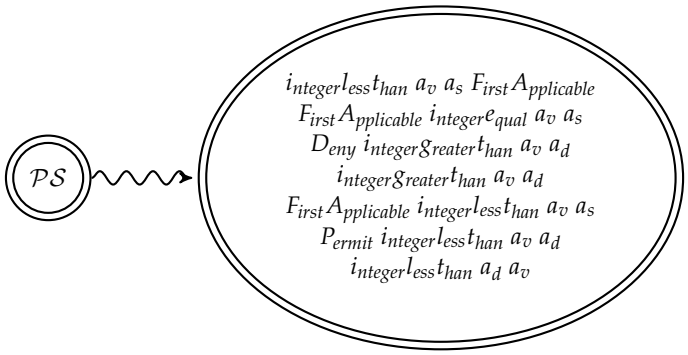
122–158 The `Rule2` element returns the `Permit` effect (line 123) in case: (i) the subject element is less than `7` as declared in the rule `Target` (lines 124–142 of Listing 1) and (ii) the subject element is less than `3` as declared in the rule `Condition` (lines 143–157 of Listing 1). This can be expressed as:

$$\boxed{\boxed{R_2}} \rightarrow \boxed{E_{ffect}} \ \boxed{T} \ C \rightarrow P_{ermit} \ \boxed{A_{nyOf}} \ C \rightarrow P_{ermit} \ \boxed{A_{llOf}} \ C \rightarrow P_{ermit} \ \boxed{M} \ C \rightarrow$$
$$P_{ermit} \ \boxed{F_{un}} \ a_v \ \boxed{A_{ttr}C_{hoice}} \ C \rightarrow P_{ermit} \ i_{nteger}l_{ess}t_{han} \ a_v \ a_d \ \boxed{C} \rightarrow$$
$$P_{ermit} \ i_{nteger}l_{ess}t_{han} \ a_v \ a_d \ \boxed{E_{X_{bool}}} \rightarrow P_{ermit} \ i_{nteger}l_{ess}t_{han} \ a_v \ a_d \ \boxed{A} \rightarrow$$

$$P_{ermit}\ i_{nteger}l_{ess}t_{han}\ a_v\ a_d\ \boxed{A}\ \boxed{E_X} \rightarrow P_{ermit}\ i_{nteger}l_{ess}t_{han}\ a_v\ a_d\ \boxed{A}\ \boxed{E_X}\ a_v \rightarrow$$
$$P_{ermit}\ i_{nteger}l_{ess}t_{han}\ a_v\ a_d\ \boxed{E_{X_{fun}}}\ a_d\ a_v \rightarrow$$

$$\boxed{\boxed{P_{ermit}\ i_{nteger}l_{ess}t_{han}\ a_v\ a_d\ i_{nteger}l_{ess}t_{han}\ a_d\ a_v}}$$

Basically, if we see the parsing procedure of this policy as an automaton, we obtain:



In addition, for this example, the parsing procedure returns a positive result, that is, the policy is fully covered by the grammar; this means that the policy is syntactically correct.

Listing 1. XACML Policy Example.

```
<PolicySet xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:
  core:schema:wd-17
  http://docs.oasis-open.org/xacml/3.0/
  xacml-core-v3-schema-wd-17.xsd"
  PolicySetId="urn:oasis:names:tc: xacml:3.0:
  conformance-test:IIIA025:policyset"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:
  policy-combining-algorithm:first-applicable"
  Version="1.0" xmlns="urn:oasis:names:tc:
  xacml:3.0:core:schema:wd-17"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:3.0:
          function:integer-less-than">
          <AttributeValue DataType="http://www.w3.org/2001/
            XMLSchema#integer">
          15</AttributeValue>
          <AttributeDesignator DataType="http://www.w3.org/2001/
            XMLSchema#integer"
            Category="urn:oasis:names:tc :xacml:3.0:
            subject-category:access-subject" MustBePresent="false"
            AttributeId="urn:oasis:names :tc:xacml:3.0:
            subject:subject-id"/>
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
  <Policy Version="1.0" RuleCombiningAlgId="urn:oasis:names:tc
    :xacml:3.0:
    rule-combining-algorithm:first-applicable"
    PolicyId="urn:oasis:names:tc:xacml:3.0:
    conformance-test:IIIA025:policy1"
    xmlns:xacml="urn:oasis:names:tc:xacml:3.0:
    core:schema:wd-17"
    xmlns="urn:oasis:names:tc:xacml:3.0:
    core:schema:wd-17">
    <Target>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:3.0:
            function:integer-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
              XMLSchema#integer">
              10</AttributeValue>
```

```
            <AttributeDesignator DataType="http://www.w3.org/2001/
              XMLSchema#integer"
              Category="urn:oasis:names:tc:xacml:3.0:
              subject-category:access-subject" MustBePresent="false"
              AttributeId="urn:oasis:names:tc:xacml:3.0:
              subject:subject-id"/>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
    <Rule RuleId="urn:oasis:names:tc:xacml:3.0:
      conformance-test:IIIA026:rule1" Effect="Deny">
      <Target>
        <AnyOf>
          <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:3.0:
              function:integer-greater-than">
              <AttributeValue DataType="http://www.w3.org/2001/
                XMLSchema#integer">
                4</AttributeValue>
              <AttributeDesignator AttributeId="urn:oasis:names:
                tc:xacml:3.0:subject:subject-id"
                DataType="http://www.w3.org/2001/
                XMLSchema#integer"
                MustBePresent="false"
                Category="urn:oasis:names:tc:xacml:3.0:
                subject-category:access-subject"/>
            </Match>
          </AllOf>
        </AnyOf>
      </Target>
      <Condition>
        <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:
          function:integer-greater-than">
          <AttributeDesignator
            AttributeId="urn:oasis:names:tc:
            xacml:3.0:conformance-test:test"
            DataType="http://www.w3.org/2001/
            XMLSchema#integer"
            MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:3.0:
            subject-category:access-subject" />
          <AttributeValue
            DataType="http://www.w3.org/2001/
            XMLSchema#integer">2</AttributeValue>
        </Apply>
      </Condition>
    </Rule>
  </Policy>
  <Policy Version="1.0" RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:
    rule-combining-algorithm:first-applicable"
    PolicyId="urn:oasis:names:tc:xacml:3.0:
    conformance-test:IIIA025:policy2"
    xmlns:xacml="urn:oasis:names:tc:xacml:3.0:
    core:schema:wd-17"
    xmlns="urn:oasis:names:tc:xacml:3.0:
    core:schema:wd-17">
    <Target>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:3.0:
            function:integer-less-than">
            <AttributeValue DataType="http://www.w3.org/2001/
              XMLSchema#integer">
              9</AttributeValue>
            <AttributeDesignator DataType="http://www.w3.org/2001/
              XMLSchema#integer"
              Category="urn:oasis:names:tc:xacml:3.0:
              subject-category:access-subject" MustBePresent="false"
              AttributeId="urn:oasis:names:tc:xacml:3.0:
              subject:subject-id"/>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
    <Rule RuleId="urn:oasis:names:tc:xacml:3.0:
      conformance-test:IIIA026:rule2" Effect="Permit">
      <Target>
        <AnyOf>
          <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:3.0:
```

```
                function:integer-less-than">
                <AttributeValue DataType="http://www.w3.org/2001/
                  XMLSchema#integer">
                  7</AttributeValue>
                <AttributeDesignator AttributeId="urn:oasis:names:tc:
                  xacml:3.0:subject:subject-id"
                  DataType="http://www.w3.org/2001/
                  XMLSchema#integer"
                  MustBePresent="false"
                  Category="urn:oasis:names:tc:xacml:3.0:
                  subject-category:access-subject"/>
              </Match>
            </AllOf>
          </AnyOf>
        </Target>
        <Condition>
          <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:
            function:integer-less-than">
            <AttributeDesignator AttributeId="urn:oasis:names:tc:
              xacml:3.0:conformance-test:test"
              DataType="http://www.w3.org/2001/
              XMLSchema#integer"
              MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:3.0:
              subject-category:access-subject"/>
            <AttributeValue DataType="http://www.w3.org/2001/
              XMLSchema#integer">
              3</AttributeValue>
          </Apply>
        </Condition>
      </Rule>
    </Policy>
</PolicySet>
```

5. Definition of the Request Structure

In this section, we provide the formalization of the XACML request information (such as attribute values, ID, and so on) necessary for being evaluated by a policy. In particular, in Figure 2, the generic request structure is represented, which contains:

- `AttributeValue` element defined by a `DataType` and any attribute.
- `Attribute` element $\mathcal{A}$ is composed by $\{\texttt{AttributeID}, \mathcal{AV}_1, \ldots, \mathcal{AV}_n\}, n \geq 1$, where $\forall i, \mathcal{AV}_i$ is an `AttributeValue` and the `AttributeID` is the identifier of the attribute.
- `Attributes` element $\mathcal{A}$ is defined by an `AttributeID`, a `Category` and a set of `Attribute` (that can also be empty), so basically:

$$\mathcal{A} = \{\texttt{AttributeID}, \texttt{Category}, \mathcal{A}_1, \ldots, \mathcal{A}_n\}, n \geq 0$$

- `Request` element: $\mathcal{R}$ has this structure: $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}, n \geq 1$ where $\forall i, \mathcal{A}_i$ is an `Attributes` element.

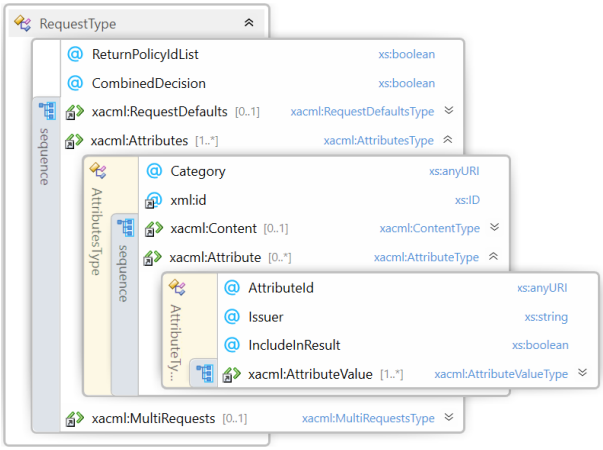


Figure 2. Request structure.

In Listings 2 and 3, two examples of requests relative to the policy of Listing 1 are provided.

Listing 2. XACML Request1 Example.

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:
  core:schema:wd-17"
  CombinedDecision="false" ReturnPolicyIdList="false">
  <Attributes Category="urn:oasis:names:tc:xacml:1.0:
    subject-category:access-subject">
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:
      subject:subject-id"
      IncludeInResult="false">
      <AttributeValue
        DataType="http://www.w3.org/2001/
        XMLSchema#integer">
        10</AttributeValue>
    </Attribute>
  </Attributes>
</Request>
```

Listing 3. XACML Request2 Example.

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:
  core:schema:wd-17"
  CombinedDecision="false" ReturnPolicyIdList="false">
  <Attributes Category="urn:oasis:names:tc:xacml:1.0:
    subject-category:access-subject">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:
      subject:subject-id"
      IncludeInResult="false">
      <AttributeValue
        DataType="http://www.w3.org/2001/
        XMLSchema#integer">
        2</AttributeValue>
    </Attribute>
  </Attributes>
</Request>
```

6. XACML-Based Coverage Definition

In this section, we detail the concepts of coverage of the `XACML 3.0` grammar elements. Considering therefore a `request` $\mathcal{R}$ and an element of `XACML 3.0` grammar, the following coverage function can be introduced.

Definition 1 (Coverage function)**.** *Let* el_{XACML} *be an* `XACML 3.0` *component and* $\mathcal{R}$ *a* `request` *element, such that* $\mathcal{R} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$

We define the function

$$cov : el_{\mathcal{XACML}} \times \mathcal{R} \rightarrow b$$

where $b \in \{T_{true}, F_{alse}\}$.

In practice, this function establishes if the request can be evaluated or not by the considered element.

Following a more intuitive way of coverage analysis, in the remainder of this section, we provide the coverage of each of the grammar elements starting from the basic ones.

6.1. Covering Match and Condition

Let $M = f_{MatchID}(v, c, a)$ be a `Match` element where $f_{MatchID}$ is an identifier of the `Match` function, v is the embedded `AttributeValue`, c is the attribute `Category` related to v, and a is an `AttributeChoice` (defined before, that can be mutually or exclusively, an `AttributeDesignator` or an `AttributeSelector`). Covering the `Match` element with a request $\mathcal{R} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ is defined as follows:

$$cov(f_{MatchID}(v,c,a),\mathcal{R}) = \begin{cases} T_{rue} & \text{if } \exists i : v',c' \in \mathcal{A}_i. \\ & f_{MatchID}(v',c',a) = T_{rue} \\ F_{alse} & \text{if } \forall i : \forall v',c' \in \mathcal{A}_i. \\ & f_{MatchID}(v',c',a) = F_{alse} \end{cases} \tag{1}$$

Note that in the definition, $f_{MatchID}$ appears twice: the former one for the syntax element of `Match` component, while the latter indicates the coverage of the $f_{MatchID}$ function.

In this case, considering the request of Listing 2 which has a subject with value 10, it is able to cover the $f_{MatchID}$ of the Policy Set of Listing 1.

Considering instead the coverage of the condition element, let $\mathcal{C}$ be a `Condition` and $\mathcal{R}$ be a request; then, it is possible to define:

$$cov(\mathcal{C},\mathcal{R}) = \begin{cases} T_{rue} \\ F_{alse} \end{cases}$$

Considering the policy of Listing 1 and the request of Listing 2, the request covers the condition expressed in the first rule with the value true because it contains a value greater than 2.

6.2. Covering AllOf, AnyOf and Target

Let $\mathcal{M}$ be a `Match`, $\mathcal{A}$ a `AllOf`, $\mathcal{E}$ a `AnyOf`, and $\mathcal{T}$ a `Target`.

Suppose that $\texttt{allof}(\mathcal{A}_{id}) : \{\mathcal{M}_1,\ldots,\mathcal{M}_n\}, n \geq 1$ is an `AllOf` element and $\forall i, \mathcal{M}_i$ is a `Match` element. The coverage of `AllOf` element $\mathcal{A}_{id}$ over a request $\mathcal{R}$ is as follows:

$$cov(\mathcal{A}_{id},\mathcal{R}) = \begin{cases} T_{rue} & \text{if } \forall i, 1 \leq i \leq n : cov(\mathcal{M}_i,\mathcal{R}) = T_{rue} \\ F_{alse} & \text{if } \exists i, 1 \leq i \leq n : cov(\mathcal{M}_i,\mathcal{R}) = F_{alse} \end{cases}$$

Suppose that $\texttt{anyof}(\mathcal{E}_{id}) : \{\mathcal{A}_1,\ldots\mathcal{A}_n\}, n \geq 1$ is an `AnyOf` element and $\forall i, \mathcal{A}_i$ is a `AllOf` element. The coverage of `AnyOf` element $\mathcal{E}_{id}$ over a `Request` $\mathcal{R}$ is defined as follows:

$$cov(\mathcal{E}_{id},\mathcal{R}) = \begin{cases} T_{rue} & \text{if } \exists i, 1 \leq i \leq n : cov(\mathcal{A}_i,\mathcal{R}) = T_{rue} \\ F_{alse} & \text{if } \forall i, 1 \leq i \leq n : cov(\mathcal{A}_i,\mathcal{R}) = F_{alse} \end{cases}$$

Suppose that $\texttt{target}(\mathcal{T}_{id}) : \{\mathcal{E}_1,\ldots,\mathcal{E}_n\}, n \geq 0$ is a `Target` element and $\forall i, \mathcal{E}_i$ is an `AnyOf` element. The coverage of `Target` $\mathcal{T}_{id}$ over a `Request` $\mathcal{R}$ is defined with the following equation:

$$cov(\mathcal{T}_{id},\mathcal{R}) = \begin{cases} T_{rue} & \text{if } (n=0) \vee (\forall i, 1 \leq i \leq n : \\ & cov(\mathcal{E}_i,\mathcal{R}) = T_{rue}) \\ F_{alse} & \text{if } \exists i, 1 \leq i \leq n : cov(\mathcal{E}_i,\mathcal{R}) = F_{alse} \end{cases}$$

Considering the policy of Listing 1, the target of *Rule1* and the request of Listing 2, the request covers the `AllOf`, the `AnyOf` and the *Target* elements of this rule with the values true. Indeed, the value 10 of the request matches the function `string equal` of the `AllOf` element, and because the `AnyOf` and the `Target` contain only an element, consequently, the request covers also these last.

6.3. Covering Rule

Let $\mathcal{R}$ be a `Rule`, $\mathcal{T}$ a `Target`, $\mathcal{C}$ a `Condition`, E an `Effect` (with possible values: `permit` and `deny`).

Suppose that $\texttt{rule}(\mathcal{R}) = \{\mathcal{T}, \mathcal{C}, E\}$. The coverage of `Rule` element $\mathcal{R}_{id}$ over a request $\mathcal{R}$ is determined as follows:

$$cov(\mathcal{R}_{id}, \mathcal{R}) = \begin{cases} T_{rue} & \text{if } \big(cov(\mathcal{T}, \mathcal{R}) = T_{rue} \ \vee \ \mathcal{T} = \epsilon\big) \wedge \\ & \big(cov(\mathcal{C}, \mathcal{R}) = T_{rue} \ \vee \ \mathcal{C} = \epsilon\big) \\ F_{alse} & \text{if } cov(\mathcal{T}, \mathcal{R}) = F_{alse} \vee \\ & \big(cov(\mathcal{T}, \mathcal{R}) = T_{rue} \wedge \\ & cov(\mathcal{C}, \mathcal{R}) = F_{alse}\big) \end{cases}$$

Considering the policy of Listing 1 and the request of Listing 2, the request covers the `Rule1` since it covers with a true value both the `Target` and `Condition` of `Rule1`.

6.4. Covering Policy

Let $\mathcal{P}$ be a `Policy`, $\mathcal{T}$ a `Target`, $\langle \mathcal{R}_1, \ldots \mathcal{R}_n \rangle$ a sequence of `Rules` and $comb_{id}$ a `Combining Algorithm`.

Suppose that $\texttt{policy}(\mathcal{P}_{id}) = \Big\{\mathcal{T}, \langle \mathcal{R}_1, \ldots \mathcal{R}_n \rangle, comb_{id}\Big\}$.

We define with $\mathbb{R}$ the coverage over a request $\mathcal{R}$ of the sequence of `Rules` :

$$\mathbb{R} = \Big\langle cov(\mathcal{R}_1, \mathcal{R}), \ldots cov(\mathcal{R}_n, \mathcal{R}) \Big\rangle$$

Then, the coverage of `Policy` $\mathcal{P}_{id}$ over a request $\mathcal{R}$ is defined as follows:

$$cov(\mathcal{P}_{id}, \mathcal{R}) = \begin{cases} T_{rue} & \text{if } cov(\mathcal{T}, \mathcal{R}) = T_{rue} \ \wedge \\ & comb_{id}(\mathbb{R}) = T_{rue} \\ F_{alse} & \text{if } cov(\mathcal{T}, \mathcal{R}) = F_{alse} \ \vee \\ & \big(cov(\mathcal{T}, \mathcal{R}) = T_{rue} \ \wedge \\ & comb_{id}(\mathbb{R}) = F_{alse}\big) \end{cases}$$

The coverage of the sequence of `Rules`, defined by $\mathbb{R}$, depends on the chosen `Combining Algorithm` $comb_{id}$. The semantic within the combining algorithm bound to the coverage of $\mathbb{R}$ over a request $\mathcal{R}$ can be formally defined is this way:

- If $comb_{id} = f_a$, then the algorithm that must be applied is the *first-applicable*:

$$f_a(\mathbb{R}) = \begin{cases} T_{rue} & \text{if } \exists i, 1 \leq i \leq n : \\ & cov(\mathcal{R}_i, \mathcal{R}) = T_{rue} \wedge \\ & \forall j, 1 \leq j < i : cov(\mathcal{R}_j, \mathcal{R}) = F_{alse} \\ F_{alse} & \text{if } \forall i, 1 \leq i \leq n : cov(\mathcal{R}_i, \mathcal{R}) = F_{alse} \end{cases}$$

- If $comb_{id} = oo_a$, then the algorithm that must be applied is the *only-one-applicable*:

$$oo_a(\mathbb{R}) = \begin{cases} T_{rue} & \text{if } \exists i, 1 \leq i \leq n : \\ & cov(\mathcal{R}_i, \mathcal{R}) = T_{rue} \wedge \\ & \big(\forall j, 1 \leq j \leq n : i \neq j \wedge \\ & cov(\mathcal{R}_j, \mathcal{R}) = F_{alse}\big) \\ F_{alse} & \text{if } \forall i, 1 \leq i \leq n : cov(\mathcal{R}_i, \mathcal{R}) = F_{alse} \end{cases}$$

- If $comb_{id} = p_o$, then the algorithm that must be applied is the *permit-overrides*:

$$p_o(\mathbb{R}) = \begin{cases} T_{rue} & \text{if } \exists i, 1 \leq i \leq n : cov(\mathcal{R}_i, \mathcal{R}) = T_{rue} \wedge \\ & E_{\mathcal{R}_i} = \texttt{permit} \\ F_{alse} & \text{if } \forall i, 1 \leq i \leq n : cov(\mathcal{R}_i, \mathcal{R}) = F_{alse} \vee \\ & \big(\exists i, 1 \leq i \leq n : cov(\mathcal{R}_i, \mathcal{R}) = T_{rue} \wedge \\ & E_{\mathcal{R}_i} = \texttt{deny}\big) \end{cases}$$

- If $comb_{id} = d_o$ then the algorithm that must be applied is the *deny-overrides*:

$$d_o(\mathbb{R}) = \begin{cases} T_{rue} & \text{if } \exists i, 1 \leq i \leq n : cov(\mathcal{R}_i, \mathcal{R}) = T_{rue} \wedge \\ & E_{\mathcal{R}_i} = \texttt{deny} \\ F_{alse} & \text{if } \forall i, 1 \leq i \leq n : cov(\mathcal{R}_i, \mathcal{R}) = F_{alse} \vee \\ & \left(\exists i, 1 \leq i \leq n : cov(\mathcal{R}_i, \mathcal{R}) = T_{rue} \wedge \right. \\ & \left. E_{\mathcal{R}_i} = \texttt{permit}\right) \end{cases}$$

Considering the policy of Listing 1 and the request of Listing 2, the request covers `Policy1`, since it covers with true value both the `Target` of the policy and the first applicable rule that is `Rule1`.

6.5. Covering PolicySet

Let $\mathcal{PS}$ be a `PolicySet`, $comb_{id}$ a `Combining Algorithm`, $\mathcal{T}$ a target, $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$ a sequence of `Policy` and `PolicySet`. Suppose that

$$\texttt{policyset}(\mathcal{PS}_{id}) = \left\{ comb_{id}, \mathcal{T}, \langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle \right\}$$

We define $\mathbb{P}$ as the coverage over a request $\mathcal{R}$ of the sequence of `Policy` and `PolicySet`:

$$\mathbb{P} = \langle cov(\mathcal{P}_1, \mathcal{R}), \ldots, cov(\mathcal{P}_n, \mathcal{R}) \rangle$$

Then, the covering of `PolicySet` $\mathcal{PS}_{id}$ is defined as follows:

$$cov(\mathcal{PS}_{id}, \mathcal{R}) = \begin{cases} T_{rue} & \text{if } cov(\mathcal{T}, \mathcal{R}) = T_{rue} \wedge \\ & comb_{id}(\mathbb{P}) = T_{rue} \\ F_{alse} & \text{if } cov(\mathcal{T}, \mathcal{R}) = F_{alse} \vee \\ & \left(cov(\mathcal{T}, \mathcal{R}) = T_{rue} \wedge \right. \\ & \left. comb_{id}(\mathbb{P}) = F_{alse}\right) \end{cases}$$

We do not formally provide here the definition of policy-combining algorithm because the semantic can be easily deduced from that already provided for the rule-combining algorithm over a request presented in the previous section.

Considering the policy of Listing 1 and the request of Listing 2, the request covers the `PolicySet`, since it covers with a true value both the `Target` of the `PolicySet` and the first applicable policy that is `Policy1`.

7. Coverage Criteria

Considering the coverage concepts introduced in the previous section, several criteria can be defined for testing purposes. In this section, we introduce some of them considering different ways in which a target and a condition element could be exercised during the requests evaluation. Specifically, some basic coverage criteria are: either having all the policy `Target` elements covered with the *True* value or having all the policy `Target` elements covered with the *True* and *False* value. Similarly, other coverage criteria are: either having all the policy `Condition` elements covered with the *True* value or having all the policy `Condition` elements covered with the *True* and *False* value. From a more formal point of view, this can be translated into:

- `Policy targets true`:
  Let $\mathcal{RQ}$ be a set of requests $\langle \mathcal{RQ}_1, \ldots \mathcal{RQ}_m \rangle$, a

$$\texttt{policyset}(\mathcal{PS}) = \{comb_{ps}, \mathcal{TPS}, \langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle\}$$

  where $comb_{ps}$ is the `Combining Algorithm`, $\mathcal{TPS}$ is the target of the policy set, and $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$ a sequence of `Policy` elements. Let each of the $\mathcal{P}_{id}$ be a $\texttt{policy}(\mathcal{P}_{id}) =$

$\left\{\mathcal{TP}_{id}, \langle \mathcal{R}_1, \ldots \mathcal{R}_h \rangle, comb_p\right\}$ where $\mathcal{TP}_{id}$ is the `Target` of the policy, $\langle \mathcal{R}_1, \ldots \mathcal{R}_h \rangle$ is the set of `Rules` and $comb_p$ is the `Combining Algorithm`. Finally, let each $\mathcal{R}_{id}$ be a $\texttt{rule}(\mathcal{R}_{id}) = \{\mathcal{TR}_{id}, \mathcal{C}, E\}$ where $\mathcal{TR}_{id}$ is the `Target` of the rule, $\mathcal{C}$ is the `Condition`, and E is the `Effect`.
The set of requests $\mathcal{RQ}$ covers the criterion of the `Policy targets true` if for each of the target elements included in the policy $\mathcal{PS}$ ($\mathcal{TPS}, \mathcal{TP}_{id}, \mathcal{TR}_{id}$) there exists at least an $\mathcal{RQ}_i$ that covers the target with the true value.

- `Policy targets true and false`.
  Let $\mathcal{RQ}$ be a set of requests $\langle \mathcal{RQ}_1, \ldots \mathcal{RQ}_m \rangle$ a

$$\texttt{policyset}(\mathcal{PS}) = \{comb_{ps}, \mathcal{TPS}, \langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle\}$$

  where $comb_{ps}$ is the `Combining Algorithm`, $\mathcal{TPS}$ is the target of the policy set and $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$ is a sequence of `Policy` elements. Let each of the $\mathcal{P}_{id}$ be a $\texttt{policy}(\mathcal{P}_{id}) = \left\{\mathcal{TP}_{id}, \langle \mathcal{R}_1, \ldots \mathcal{R}_h \rangle, comb_p\right\}$ where $\mathcal{TP}_{id}$ is the `Target` of the policy, $\langle \mathcal{R}_1, \ldots \mathcal{R}_h \rangle$ is the set of `Rules` and $comb_p$ is the `Combining Algorithm`. Finally, let each $\mathcal{R}_{id}$ be a $\texttt{rule}(\mathcal{R}_{id}) = \{\mathcal{TR}_{id}, \mathcal{C}, E\}$ where $\mathcal{TC}_{id}$ is the `Target` of the rule, $\mathcal{C}$ is the `Condition`, and E is the `Effect`.
  The set of requests $\mathcal{RQ}$ covers the criterion of the `Policy targets true and false` if for each of the target element included in the policy $\mathcal{PS}$ ($\mathcal{TPS}, \mathcal{TP}_{id}, \mathcal{TR}_{id}$), there exists at least an $\mathcal{RQ}_i$ that covers the target with the true value and at least an $\mathcal{RQ}_j$ that covers the target with the false value.
  According to the target coverage defined in the previous section, if a `Target` element is empty, it is always covered with T_{true}. From a practical point of view, the *True* evaluation of a `Target` element requires a request that makes at least one element `AnyOf` to *True*; in case of the `AnyOf` element of the `AllOf` elements, a request is necessary that makes all the `Match` elements *True* (in our analysis, the `Match` elements are the leaves of the tree with the `Policy` or `PolicySet` component as the root).
  In a similar way, two additional criteria that refine the previous ones can be defined. They focus on the evaluation of the condition and can be formulated in the following.
- `Policy conditions true`:
  Let $\mathcal{RQ}$ be a set of requests $\langle \mathcal{RQ}_1, \ldots \mathcal{RQ}_m \rangle$, a

$$\texttt{policyset}(\mathcal{PS}) = \{comb_{ps}, \mathcal{TPS}, \langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle\}$$

  where $comb_{ps}$ is the `Combining Algorithm`, $\mathcal{TPS}$ is the target of the policy set and $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$ is a sequence of `Policy` elements. Let each of the $\mathcal{P}_{id}$ be a $\texttt{policy}(\mathcal{P}_{id}) = \left\{\mathcal{TP}_{id}, \langle \mathcal{R}_1, \ldots \mathcal{R}_h \rangle, comb_p\right\}$ where $\mathcal{TP}_{id}$ is the `Target` of the policy, $\langle \mathcal{R}_1, \ldots \mathcal{R}_h \rangle$ is the set of `Rules` and $comb_p$ is the `Combining Algorithm`. Finally, let each of $\mathcal{R}_{id}$ be a $\texttt{rule}(\mathcal{R}_{id}) = \{\mathcal{TR}, \mathcal{C}_{id}, E\}$ where $\mathcal{TR}$ is the `Target` of the rule, $\mathcal{C}_{id}$ is the `Condition` and E is the `Effect`.
  The set of requests $\mathcal{RQ}$ covers the criterion of the `Policy conditions true` if for each of the condition elements included in the rule $\mathcal{R}$ ($\mathcal{TR}, \mathcal{C}_{id}, E$), there exists at least an $\mathcal{RQ}_i$ that covers the condition with the true value.
- `Policy conditions true and false`:
  Let $\mathcal{RQ}$ be a set of requests $\langle \mathcal{RQ}_1, \ldots \mathcal{RQ}_m \rangle$, a

$$\texttt{policyset}(\mathcal{PS}) = \{comb_{ps}, \mathcal{TPS}, \langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle\}$$

  where $comb_{ps}$ is the `Combining Algorithm`, $\mathcal{TPS}$ is the target of the policy set and $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$ is a sequence of `Policy` elements. Let each of the $\mathcal{P}_{id}$ be a $\texttt{policy}(\mathcal{P}_{id}) = \left\{\mathcal{TP}_{id}, \langle \mathcal{R}_1, \ldots \mathcal{R}_h \rangle, comb_p\right\}$ where $\mathcal{TP}_{id}$ is the `Target` of the policy, $\langle \mathcal{R}_1, \ldots \mathcal{R}_h \rangle$ is the set of `Rules` and $comb_p$ is the `Combining Algorithm`. Finally, let each $\mathcal{R}_{id}$ be a

`rule`$(\mathcal{R}_{id}) = \{\mathcal{TR}, \mathcal{C}_{id}, E\}$ where $\mathcal{TR}$ is the `Target` of the rule, $\mathcal{C}_{id}$ is the `Condition`, and E is the `Effect`.

The set of requests $\mathcal{RQ}$ covers the criterion of the `Policy conditions true and false` if for each of the condition elements included in the rule $\mathcal{R}$ $(\mathcal{TR}, \mathcal{C}_{id}, E)$, there exists at least an $\mathcal{RQ}_i$ that covers the condition with the true value and at least an $\mathcal{RQ}_j$ that covers the condition with the false value.

7.1. Application Example of Coverage Criteria

In this section, as an example, we show the application of the coverage criteria defined in Section 7 to the policy of Listing 1 described in Section 4.3, and we provide examples of test suites able to satisfy the different proposed criteria.

7.1.1. Policy Target True

Given a test suite $TS_1 = \{RQ_1\}$, where RQ_1 is the request of Listing 2, we would like to measure its coverage considering the *Policy target true* criterion. As in Listing 2, the request contains a subject having a value equal to 10. The evaluation of the request on the policy of Listing 1 provides the following results: RQ_1 covers with *True* value the `Target` of the `PolicySet`, the `Target` of `Policy1` and the `Target` of the `Rule1`.

Indeed, because the policy-combining algorithm is `FirstApplicable`, i.e., it returns the result of the first applicable policy, neither `Policy2` nor `Rule2` are evaluated with RQ_1.

Consequently, the overall coverage measure of the *Policy targets true* criterion for TS_1 is 60%, because the policy of Listing 1 contains five targets and RQ_1 covers only three of them.

In order to increase the coverage and reach 100%, TS_1 should be enriched with a request able to reach the target of `Policy2`. From a practical point of view, this is possible by executing a request that makes the evaluation of the `Policy1` not applicable and triggers the evaluation of `Policy2`. An example of such a request is provided in Listing 3, which contains a subject having a value equal to 2. Indeed, this request is able to pass the `Target` of the `PolicySet`, makes the evaluation of the `Policy1` not applicable, and triggers the evaluation both of the `Target` of `Policy2` and `Rule2`.

Consequently, given a test suite $TS_2 = \{RQ_1, RQ_2\}$ where RQ_1 and RQ_2 are the requests of Listing 2 and Listing 3, respectively, the overall coverage measure of the *Policy targets true* criterion for TS_2 is 100% because RQ_2 is able to evaluate with a *True* value the `Targets` of the `PolicySet`, `Policy2` and `Rule2`.

7.1.2. Policy Target True and False

Given the test suite $TS_2 = \{RQ_1, RQ_2\}$ where RQ_1 and RQ_2 are the requests of Listing 2 and Listing 3, respectively, we would like to measure its coverage considering the *Policy target true and false* criterion.

In this case, as already described in the previous section, RQ_1 and RQ_2 are able to evaluate with *True* value the `Targets` of the `PolicySet`, `Policy1`, `Policy2`, `Rule1` and `Rule2`. Additionally, the request R_2 evaluates with *False* value the target of `Policy1`.

Consequently, the overall coverage measure of the *Policy targets true and false* criterion for TS_2 is 60%, because the policy of Listing 1 contains five targets, each one to be evaluated to *True* and *False* value, and TS_2 covers only six of them.

In order to increase the coverage and try to reach 100%, TS_2 should be enriched with additional requests. For the aim of simplicity, we report in Table 5 an example of the required requests. In particular, we considered RQ_3 having a subject value equal for instance to 16; (ii) RQ_4 with a subject value equal for instance to 11; (iii) RQ_5 with a subject value equal for instance to 8.

Consequently, given a test suite $TS_3 = TS_2 \bigcup \{RQ_3, RQ_4, RQ_5\}$, we would like to measure its coverage considering the *Policy target true and false* criterion.

In this case

- RQ_3 is able to evaluate with *False* value the `Target` of the `PolicySet`;

- RQ_4 is able to evaluate with *False* value the `Target` of `Policy2`;
- RQ_5 is able to reach the evaluation with *True* value of the `Target` of `Policy2` and then trigger the evaluation of the `Target` of the `Rule2` with *False* value.

Consequently, the overall coverage measure of the *Policy targets true and false* criterion for TS_3 is 90%.

Note that it is not possible to reach 100% coverage of *Policy targets true and false* criterion, since it is not possible to have the coverage of the `Target` of the `Rule1` with *False* value. This is because in order to trigger the evaluation of `Rule1`, we need a request with a subject equal to 10 that is able to evaluate with *True* value the `Target` of `Policy1`; then, this subject value is always greater than 4.

7.1.3. Policy Condition True

Given the test suite $TS_2 = \{RQ_1, RQ_2\}$ where RQ_1 and RQ_2 are the requests of Listing 2 and Listing 3, respectively, we would like to measure its coverage considering the *Policy condition true* criterion.

In this case, as already described in the previous section, RQ_1 and RQ_2 are able to evaluate with *True* value the `Conditions` of `Rule1` and `Rule2`. Indeed, RQ_1 is able to evaluate with *True* value the condition of `Rule1`, whereas RQ_2 is able to evaluate with *True* value the condition of `Rule2`.

Consequently, the test suite TS_2 is also able to reach 100% coverage of `Policy conditions true` criterion.

7.1.4. Policy Condition True and False

Given the test suite $TS_4 = TS_2 \bigcup \{RQ_6\}$ where RQ_6 is the request of Table 5, we would like to measure its coverage considering the *Policy condition true and false* criterion.

In this case, as already described in the previous section, RQ_1, RQ_2 and RQ_6 are able to reach 75% of coverage of `Policy conditions true and false` criterion, since RQ_6 is able to cover with *False* value the condition of `Rule2`.

Note that, again, it is not possible to reach a 100% coverage measure of `Policy conditions true and false` criterion, since to trigger the evaluation of condition of `Rule1`, a subject equal to 10 is needed to evaluate to *True* value the `Target` of `Policy1`. This subject value makes always true the condition of `Rule1`.

Table 5. Subject value for each request.

Request	RQ_1	RQ_2	RQ_3	RQ_4	RQ_5	RQ_6
Subject Value	10	2	16	11	8	4

8. Discussion and Conclusions

In this paper, we provided a formal definition of XACML syntax and semantics and defined some coverage concepts useful for verifiability and testability purposes. In particular, we revised and extended the existing definition of the abstract syntax and semantics of XACML 3.0 standard, defined a formal specification of the coverage process of the elements of the XACML policy and provided some coverage criteria useful to assess the XACML policy.

The proposed coverage criteria can have different practical implications for improving the verifiability and testability of the policy. Indeed, from a testing point of view, the policy coverage measure can describe the degree to which the policy has been exercised by a given test suite. Reaching high coverage, measured as a percentage, can increase the chance of detecting possible weaknesses or security flaws in the considered policy code. Moreover, the analysis of the policy elements that have not been covered may suggest possible improvements of the original test suite, so to increase the overall fault detection effectiveness of the test suite itself. This has been evidenced also by the application example proposed in this paper. Even if it is very simple, the application of the coverage criteria of

Target true and false and *Condition true and false* evidenced an infeasible path in the policy specification. This does not per se represent a security flaw; however, it highlights an inaccuracy in the policy writing that should be avoided. Another practical implication of the proposed coverage criteria is the possibility of exploiting the coverage measure to reduce (or select), from a given test suite, only those test cases that have an impact on the defined coverage criteria. This is specifically important in case of regression testing because the test effort may be dedicated just to run the test cases able to maximize the coverage measure.

Finally, the proposals of this paper can also be used for the definition of test suite generation methodologies that target the 100% coverage of a specific criterion. As from the application example, the coverage of a specific target and/or condition depends on the evaluation of the previous targets and/or conditions. Thus, as a basic proposal for a test case generation algorithm able to force the execution of a specific rule, it should be considered that: it is first necessary to generate the attributes that make the `Target` element of the `PolicySet` true; successively to generate the attributes that make the `Target` of the current `Policy` true and those that make the previous `Policy` elements false; finally, to generate the attributes that make the `Target` of the `Rule` element true and the `Targets` of the previous `Rules` elements false.

However, many different metrics can be used to calculate policy coverage. In this paper, we provide some basic ones focusing on the evaluation of the targets and the conditions that are common critical points for most of the policies. As a future work, we would like to provide more additional specific coverage criteria as well as to perform an accurate comparison of their fault detection effectiveness so to better guide testing efforts. We are also developing ad hoc test case generation algorithms able to target the coverage criteria proposed in this paper.

Author Contributions: Conceptualization, C.C., F.L. and E.M.; methodology, C.C., F.L. and E.M.; formal analysis, C.C., F.L. and E.M.; writing—original draft preparation, F.L. and E.M.; writing—review and editing, C.C., F.L. and E.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the European Commission, under the projects H2020 CyberSec4Europe grant number 830929 and H2020 BIECO grant number 952702.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. AlMedires, M.; AlMaiah, M. Cybersecurity in Industrial Control System (ICS). In Proceedings of the International Conference on Information Technology (ICIT), Amman, Jordan, 14–15 July 2021; pp. 640–647.
2. Hu, V.C.; Kuhn, D.R.; Ferraiolo, D.F. Attribute-based access control. *Computer* **2015**, *48*, 85–88. [CrossRef]
3. Han, W.; Lei, C. A survey on policy languages in network and security management. *Comput. Netw.* **2012**, *56*, 477–489. [CrossRef]
4. Rissanen, E. eXtensible Access Control Markup Language (XACML) Version 3.0 OASIS Standard. 2013, Volume 33, p. 110. Available online: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-corespec-os-en.html (accessed on 20 January 2022).
5. Riad, K.; Cheng, J. Adaptive XACML access policies for heterogeneous distributed IoT environments. *Inf. Sci.* **2021**, *548*, 135–152. [CrossRef]
6. Ravidas, S.; Lekidis, A.; Paci, F.; Zannone, N. Access control in Internet-of-Things: A survey. *J. Netw. Comput. Appl.* **2019**, *144*, 79–101. [CrossRef]
7. Lonetti, F.; Marchetti, E. Issues and Challenges of Access Control in the Cloud. In Proceedings of the WEBIST, Seville, Spain, 18–20 September 2018; pp. 261–268.
8. Daoudagh, S.; Lonetti, F.; Marchetti, E. Continuous Development and Testing of Access and Usage Control: A Systematic Literature Review. In Proceedings of the ESSE 2020: 2020 European Symposium on Software Engineering, Rome, Italy, 6–8 November 2020; pp. 51–59. [CrossRef]

9. Martin, E. Automated test generation for access control policies. In Proceedings of the November Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, Portland, OR, USA, 22–26 October 2006; pp. 752–753. [CrossRef]
10. Bertolino, A.; Daoudagh, S.; Lonetti, F.; Marchetti, E. Automatic XACML Requests Generation for Policy Testing. In Proceedings of the Fifth IEEE International Conference on Software Testing, Verification and Validation, Montreal, QC, Canada, 17–21 April 2012; pp. 842–849. [CrossRef]
11. Limaye, S.; Zhang, Y. Combining algorithm based data flow testing approach for XACML. In Proceedings of the Third ACM Workshop on Attribute-Based Access Control, Tempe, AZ, USA, 21 March 2018; pp. 25–31.
12. Xu, D.; Shrestha, R.; Shen, N. Automated coverage-based testing of XACML policies. In Proceedings of the 23nd ACM Symposium on Access Control Models and Technologies, Indianapolis, IN, USA, 13–15 June 2018; pp. 3–14.
13. Daoudagh, S.; Lonetti, F.; Marchetti, E. XACMET: XACML testing & modeling. *Softw. Qual. J.* **2020**, *28*, 249–282.
14. Pezzè, M.; Young, M. *Software Testing and Analysis—Process, Principles and Techniques*; Wiley: Hoboken, NJ, USA, 2007.
15. Rothermel, G.; Harrold, M.J.; Ostrin, J.; Hong, C. An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites. In Proceedings of the International Conference on Software Maintenance, Bethesda, MD, USA, 20 November 1998; pp. 34–43. [CrossRef]
16. Ramli, C.D.P.K.; Nielson, H.R.; Nielson, F. The logic of XACML. *Sci. Comput. Program.* **2014**, *83*, 80–105. [CrossRef]
17. Masi, M.; Pugliese, R.; Tiezzi, F. Formalisation and Implementation of the XACML Access Control Mechanism. *ESSoS* **2012**, *7159*, 60–74.
18. Margheri, A.; Pugliese, R.; Tiezzi, F. On Properties of Policy-Based Specifications. *arXiv* **2015**, arXiv:1508.03903.
19. Vijayalakshmi, K.; Jayalakshmi, V. A priority-based approach for detection of anomalies in ABAC policies using clustering technique. In Proceedings of the 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 11–13 March 2020; pp. 897–903.
20. Mejri, M.; Yahyaoui, H.; Mourad, A.; Chehab, M. A rewriting system for the assessment of XACML policies relationship. *Comput. Secur.* **2020**, *97*, 101957. [CrossRef]
21. Bundy, A.; Wallen, L. Context-free grammar. In *Catalogue of Artificial Intelligence Tools*; Springer: Berlin/Heidelberg, Germany, 1984; pp. 22–23.
22. The Margrave Policy Analyzer. Available online: http://www.margrave-tool.org/ (accessed on 20 February 2022).
23. Zhao, C.; Heilili, N.; Liu, S.; Lin, Z. Representation and reasoning on RBAC: A description logic approach. In Proceedings of the ICTAC, Hanoi, Vietnam, 17–21 October 2005; Volume 3722, pp. 381–393.
24. Kolovski, V.; Hendler, J.; Parsia, B. Analyzing Web Access Control Policies. In Proceedings of the 16th International Conference on World Wide Web, Banff, AB, Canada, 8–12 May 2007; pp. 677–686. [CrossRef]
25. Jackson, D. Alloy: A lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.* **2002**, *11*, 256–290. [CrossRef]
26. Bryans, J. Reasoning about XACML policies using CSP. In Proceedings of the 2005 Workshop on Secure Web Services, Fairfax, VA, USA, 11 November 2005; pp. 28–35.
27. Xu, D.; Zhang, Y.; Shen, N. Formalizing semantic differences between combining algorithms in XACML 3.0 policies. In Proceedings of the 2015 IEEE International Conference on Software Quality, Reliability and Security, Vancouver, BC, Canada, 3–5 August 2015; pp. 163–172.
28. Tout, H.; Mourad, A.; Talhi, C.; Otrok, H.; Yahyaoui, H. Model-driven specification and design-level analysis of XACML policies. In Proceedings of the Second International Conference on Next Generation Computing and Communication Technologies, Dubai, United Arab Emirates, 22–23 April 2015.
29. Yang, S.; Tan, C. Detection of Conflicts between Resource Authorization Rules in Extensible Access Control Markup Language Based on Dynamic Description Logic. *Ing. Syst. d'Inf.* **2020**, *25*, 285–294. [CrossRef]
30. Masoumzadeh, A.; Narendran, P.; Iyer, P. Towards a Theory for Semantics and Expressiveness Analysis of Rule-Based Access Control Models. In Proceedings of the 26th ACM Symposium on Access Control Models and Technologies, SACMAT'21, Virtual, Spain, 16–18 June 2021; pp. 33–43. [CrossRef]
31. Kashmar, N.; Adda, M.; Atieh, M.; Ibrahim, H. Access control metamodel for policy specification and enforcement: From conception to formalization. *Procedia Comput. Sci.* **2021**, *184*, 887–892. [CrossRef]
32. Kashmar, N.; Adda, M.; Ibrahim, H. HEAD Access Control Metamodel: Distinct Design, Advanced Features, and New Opportunities. *J. Cybersecur. Priv.* **2022**, *2*, 42–64. [CrossRef]
33. Hughes, G.; Bultan, T. Automated verification of access control policies using a sat solver. *Int. J. Softw. Tools Technol. Transf.* **2008**, *10*, 503–520. [CrossRef]
34. Shahid, M.; Ibrahim, S.; Mahrin, M.N. A Study on Test Coverage in Software Testing. In Proceedings of the International Conference on Telecommunication Technology and Applications, Syedey, Australia, 13 May 2011; IACSIT Press: Singapore, 2011; pp. 207–215.
35. Zhu, H.; Hall, P.A.; May, J.H. Software unit test coverage and adequacy. *ACM Comput. Surv.* **1997**, *29*, 366–427. [CrossRef]
36. Kaur, A.; Goyal, S. A genetic algorithm for regression test case prioritization using code coverage. *Int. J. Comput. Sci. Eng.* **2011**, *3*, 1839–1847.
37. Pradhan, S.; Ray, M.; Patnaik, S. Coverage criteria for state-based testing: A systematic review. *Int. J. Inf. Technol. Proj. Manag.* **2019**, *10*, 1–20. [CrossRef]

38. Martin, E.; Xie, T.; Yu, T. Defining and measuring policy coverage in testing access control policies. In Proceedings of the International Conference on Information and Communications Security, Raleigh, NC, USA, 4–7 December 2006; pp. 139–158.
39. Bertolino, A.; Le Traon, Y.; Lonetti, F.; Marchetti, E.; Mouelhi, T. Coverage-based test cases selection for XACML policies. In Proceedings of the ICST Workshops, Cleveland, OH, USA, 31 March–4 April 2014; pp. 12–21.
40. Martin, E.; Xie, T. Automated Test Generation for Access Control Policies via Change-Impact Analysis. In Proceedings of the Third International Workshop on Software Engineering for Secure Systems, Minneapolis, MN, USA, 20–26 May 2007. [CrossRef]
41. Hwang, J.; Xie, T.; El Kateb, D.; Mouelhi, T.; Le Traon, Y. Selection of regression system tests for security policy evolution. In Proceedings of the ASE, Essen, Germany, 3–7 September 2012; pp. 266–269.
42. Lonetti, F.; Marchetti, E. On-line tracing of XACML-based policy coverage criteria. *IET Softw.* **2018**, *12*, 480–488. [CrossRef]
43. Jiang, J.; Chirkova, R.; Doyle, J.; Rosenthal, A. Towards greater expressiveness, flexibility, and uniformity in access control. In Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, Indianapolis, IN, USA, 13–15 June 2018; pp. 217–219.