Diploma Thesis

ANALYZING EXTENSIBLE ACCESS CONTROL MARKUP LANGUAGE POLICIES USING ANWSER SET PROGRAMMING

Vladimiros Pavlos Semertsidis

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

May 2022

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

ANALYZING EXTENSIBLE ACCESS CONTROL MARKUP LANGUAGE POLICIES USING ANWSER SET PROGRAMMING

Vladimiros Pavlos Semertsidis

Supervisor

Yannis Dimopoulos

Thesis submitted in partial fulfilment of the requirements for the award of degree of bachelor's in computer science at University of Cyprus

May 2022

Acknowledgements

Firstly, I would like to say that I am extremely grateful to my supervisor, Dr. Yannis Dimopoulos for his constant support and guidance that has played a significant role in the completion of this thesis. His experience and knowledge of the field have helped me move forward even in times of doubt.

I am also grateful to the Computer Science Department of the University of Cyprus for providing me with the knowledge that served as a fundamental base for the implementation of this thesis.

Finally, I would like to thank my family and friends for being supportive and encouraging me during these very important and stressful past four years of my life. Their belief in me and my abilities has been my greatest motive to succeed.

Abstract

In a digitalised world where most systems, organizations and businesses have adopted the use web applications, the need for protection of digitality stored information within these applications is greater than ever. A widely implemented way to provide such protection are web access control policies. These access control policies are seen today as one of the best approaches to making web applications secure because they allow authorizations to define who has authorization over what resources and under which conditions in a simple manner. But despite their usefulness, they have problems. They tend to expand in size and complexity as the applications that use them grow in complexity as well, which makes it challenging for developers and analysers to manually detect errors, anomalies, and vulnerabilities within them. In other words, there is a demand for an effective and efficient tool that will help with policy analysis.

In this thesis we present and extended XACBench, a partially implemented policy analysis tool which takes a policy written in XACML (a widely adopted attribute-based access control language) and translates it into an equivalent set of ASP programs. Where ASP is a declarative language for solving search problems which when paired with an ASP solver can help with query analysis and property analysis of access policies.

Our extended version of this tool generates a more accurate ASP translation of XACML 3.0 policies by taking into account its expressiveness regarding arbitrary attribute types. We also propose a more efficient way to perform query analysis of access policies by defining a variation set of ASP programs which led to an average speed up of 1.2. Finally, we introduce a new policy property called hierarchy which gives us insight into the authorization hierarchies of each attribute type value within a policy.

Contents

Chapter 1 Introduction to XACML and ASP 1

1.1 Motivation 2

1.2 The eXtensible Access Control Markup Language (XACML) 3

1.2.1 Introduction to XACML 3

1.2.2 Essential requirements of access policy languages 3

1.2.3 Essential terms of access policy languages 4

1.2.4 The XACML access authorization process 5

1.2.5 XACML 2.0 vs XACML 3.0 6

1.2.6 XACML 3.0 abstract syntax 7

1.2.6.1 Introduction 7

1.2.6.2 Target element 8

1.2.6.3 AnyOf element 9

1.2.6.4 AllOf element 10

1.2.6.5 Match element 10

1.2.6.6 Effect element 11

1.2.6.7 Condition element 11

1.2.6.8 Rule element 12

1.2.6.9 Policy element 13

1.2.6.10 Policy Set element 14

1.2.6.11 Policy/Rule Combining algorithm element 14

1.2.6.11.1 Deny-overrides 15

1.2.6.11.2 Permit-overrides 15

1.2.6.11.3 First applicable 16

1.2.6.11.4 Only-one-applicable 16

1.2.6.12 Request element 17

1.3 Answer Set Programming (ASP) 18

1.3.1 Introduction to ASP 18

1.3.1.1 ASP syntax

1.3.1.2 Answer Sets

1.3.2 ASP workflow

1.3.2.1 ASP example 20
1.3.3 Clingo 22

Chapter 2 XACML to ASP translation and XACBench 23
2.1 Motivation 24
2.2 Proposed framework and prototype tool for translation and analysis of XACML policies 25
2.2.1 Framework for policy translation and analysis 25
2.2.2 The XACBench prototype tool for policy translation 26
2.3 ASP programs of the top framework layer 27
2.3.1 Mapping policy dependent rules 27
2.3.2 Mapping policy independent rules 30
2.3.3 Mapping combining algorithms 38
2.3.3.1 Permit overrides 38
2.3.3.2 Deny overrides 41
2.3.3.3 First applicable 43
2.3.3.4 Only one applicable 45
2.3.4 Mapping requests 46
2.4 ASP programs of the middle framework layer 47
2.4.1 Reachability 48
2.4.2 Usefulness 49
2.4.3 Total redundancy 50
2.4.4 Completeness 52
2.4.5 Isomorphism 52
2.4.6 Intra-policy anomalies 54
2.4.7 Inter-policy anomalies 56
2.5 ASP request generator 59

Chapter 3 Modifying XACBench 61
3.1 Introduction 61
3.2 Arbitrary attribute types of XACML 3.0 62
3.3 Conditions of XACML 3.0 64
3.4 Automatic target matching 69
3.5 Reducing atom arity for query analysis 71
3.6 Authorization hierarchy 74

Chapter 4 Experiments and Results **80**

4.1 Introduction 80

4.1.1 Experimental environment 81

4.2 Policy property analysis 81

4.2.1 Effectiveness 81

4.2.1.1 Results 82

4.2.2 Efficiency 83

4.2.2.1 Results 85

4.3 Efficiency of variation ASP program for query analysis 87

4.3.1 Results 88

4.4 Hierarchy 89

Chapter 5 Conclusions **92**

5.1 Summary 92

5.2 Limitations 93

5.3 Future work 93

References **95**

Appendix A **97**

Appendix B **98**

Appendix C **99**

Appendix D **100**

Chapter 1

Introduction to XACML and ASP

1.1 Motivation 2

1.2 The eXtensible Access Control Markup Language (XACML) 3

1.2.1 Introduction to XACML 3

1.2.2 Essential requirements of access policy languages 3

1.2.3 Essential terms of access policy languages 4

1.2.4 The XACML access authorization process 5

1.2.5 XACML 2.0 vs XACML 3.0 6

1.2.6 XACML 3.0 abstract syntax 7

1.2.6.1 Introduction 7

1.2.6.2 Target element 8

1.2.6.3 AnyOf element 9

1.2.6.4 AllOf element 10

1.2.6.5 Match element 10

1.2.6.6 Effect element 11

1.2.6.7 Condition element 11

1.2.6.8 Rule element 12

1.2.6.9 Policy element 13

1.2.6.10 Policy Set element 14

1.2.6.11 Policy/Rule Combining algorithm element 14

1.2.6.11.1 Deny-overrides 15

1.2.6.11.2 Permit-overrides 15

1.2.6.11.3 First-applicable 16

1.2.6.11.4 Only-one-applicable 16

1.2.6.12 Request element 17

1.3 Answer Set Programming (ASP) 18
1.3.1 Introduction to ASP 18
1.3.1.1 ASP syntax
1.3.1.2 Answer Sets
1.3.2 ASP workflow
1.3.2.1 ASP example 20
1.3.3 Clingo 22

1.1 Motivation

As we all know, we live in a digitalised world where most systems, organizations and businesses have adopted the use of web applications. This new and rapidly changing world has put a lot of pressure on corporate and government executives to protect digital information and assets of the organizations and their customers. That is why web access control policies have been widely adopted by many organizations. The general idea behind access control policies is that they provide rules and guidelines structuring who can access data and resources at an organization.

Although access control policies are seen as a good approach to making web applications secure, they seem to have some issues. One of them is their tendency to expand in size and complexity as the applications that implement them grow in complexity as well. Another issue is that many organizations that are divided into subdivisions can have their own set of policies which at sometimes need to be combined to come to a decision regarding access to a common asset. These issues make it difficult for developers and analysts of access control policies to manually analyse and detect behaviour, errors, anomalies, and vulnerabilities.

That is why, in this thesis we contribute towards overcoming these issues by modifying an already existing tool that will provide an efficient and effective way to verify a large variety of properties of access control policies. More specifically, we will be working with XACML, a widely adopted attribute-based access control language, answer set programming (ASP), a declarative language for solving search problems that has recently seen many advancements, and a predefined syntax for mapping XACML into ASP. The expanded tool will take XACML policies as input and translate them, using a mostly predefined syntax, to a set of ASP

equivalent programs. These programs will then be given to Clingo, an ASP system that grounds and solves logic programs, which will in turn verify specific properties of the access policies.

1.2 The eXtensible Access Control Markup Language (XACML)

1.2.1 Introduction to XACML

The Organization for the Advancement of Structured Information Standards (OASIS) has developed the eXtensible Access Control Markup Language (XACML) to define security policies, request context, and response context statements (all written in XML) [2]. XACML provides a rich data model for the specification of complicated conditions, that is why it has become one of the most standardized options for modelling access control policies for many web applications.

1.2.2 Essential requirements of access policy languages

A policy language for expressing information system security must [2]:

- provide a way to combine rules and policies into a single Policy Set that can be applied to a particular decision request.
- provide a way to define a process in which rules and policies are combined.
- provide a way to handle multiple subjects acting in different capacities.
- provide a way for deciding whether to provide authorization based on arbitrary attributes of the world (i.e., subject, resource, environment) that can take multiple values.
- provide a way to make authorization decisions based on the contents of an information resource.
- provide a way to perform logical and mathematical operations on arbitrary attributes of the world.
- provide a way for dealing with a dispersed set of policy components while abstracting the process for finding, obtaining, and authenticating the policy components.
- provide a way to quickly determine whether a policy applies to a particular action based on the values of the arbitrary attributes of the world.

- provide a way to specify a set of actions that must be performed in conjunction with policy enforcement.
- provide a high-level representation of the application environment so that the policy developers can avoid getting into details.

1.2.3 Essential terms of access policy languages

Some important definitions of access control [2]:

Action – an operation on a resource
Access - performing an action
Access control - authorizing access based on a policy or Policy Set
Attribute – arbitrary description of the world (i.e., subject, resource, environment, action, role)
Named attribute/attribute type value – an instance of an attribute, determined by the attribute name and type
Predicate - a statement about attributes
Conjunctive sequence - a sequence of predicates combined using the logical 'AND' operation
Disjunctive sequence - a sequence of predicates combined using the logical 'OR' operation
Condition - an expression of predicates. A function that evaluates to "True", "False" or "Indeterminate"
Effect - the consequence of a satisfied rule. The values it can take are either "Permit" or "Deny".
Decision request - a PEP's request sent to a PDP to make an authorisation decision
Target – contains a set of decision requests, identified by definitions for attributes that a rule, policy, or Policy Set must evaluate
Rule - contains target, an effect, and a condition
Rule-combining algorithm – combines decisions of multiple rules
Policy – contains a set of rules, a rule-combining algorithm, and a target
Policy-combining algorithm - combines decisions of multiple policies
Policy Set – contains a set of policies, other Policy Sets, a policy-combining algorithm, and a target
Applicable policy - The set of policies and Policy Sets that governs access for a specific decision request
Decision - the result of evaluating a rule, policy or Policy Set
Authorization decision - the result of evaluating applicable policy, returned by the PDP to the PEP. A function that evaluates to "Permit", "Deny", "Indeterminate" or "NotApplicable"

Advice - a supplementary piece of information in a policy or Policy Set which is provided to the PEP with the decision of the PDP

Obligation - a operation specified in a rule, policy or Policy Set that should be performed by the PEP in conjunction with the enforcement of an authorization decision

Context - a canonical representation of a decision request and an authorization decision

Policy information point (PIP) – system component that contains attribute values

Context handler – system component that takes a decision request written in a native format of some authority and translates it into a XACML format, and the reverse process for the response

Policy administration point (PAP) - system component that defines a policy or Policy Set

Policy decision point (PDP) – system component that evaluates applicable policy and renders an authorization decision

Policy enforcement point (PEP) - system component that defines decision requests and enforces authorization decisions

1.2.4 The XACML access authorization process

Note: In this thesis we see attribute as arbitrary descriptions of the world instead of grouping them into action, resource, environment, and action. Also, Advices and Obligations of XACML are not used in the policy analysis framework of this thesis, thus we do not cover them in such detail.

As shown in Figure 1.1, initially an authority defines policies within the PAP (1). In other words, an organisation defines its authorization scheme using XACML. These policies or Policy Sets represent the complete policy for a specified target which are later used by the PDP. Then some access requester sends an access request in a native format to the PEP (2), which forwards the request for access to the context handler in its native request format (3). The context handler, which communicates with the PIP to access different attributes, translates the request which is in native format into a request in XACML format and the sends it to the PDP (4). The PDP might need to request additional attributes from the context handler (5) which in turn requests the attributes from PIP (6). If so, the PIP obtains the requested attributes from each attribute's relative domain. Figure 1.1 only shows domains for attribute types: subject, environment, and resource and then returns them to the context handler (7-9). The context handler in turn returns the attributes to the PDP (10), which then evaluates the policy using the attributes it received and the policies defined in PAP with which it communicates. Following

the evaluation, the PDP returns the response context with the authorization decision to the context handler (11). Finally, the context handler translates the response context from the XACML format to the native response format and sends it to the PEP (12), which either authorizes or denies access to the resource.

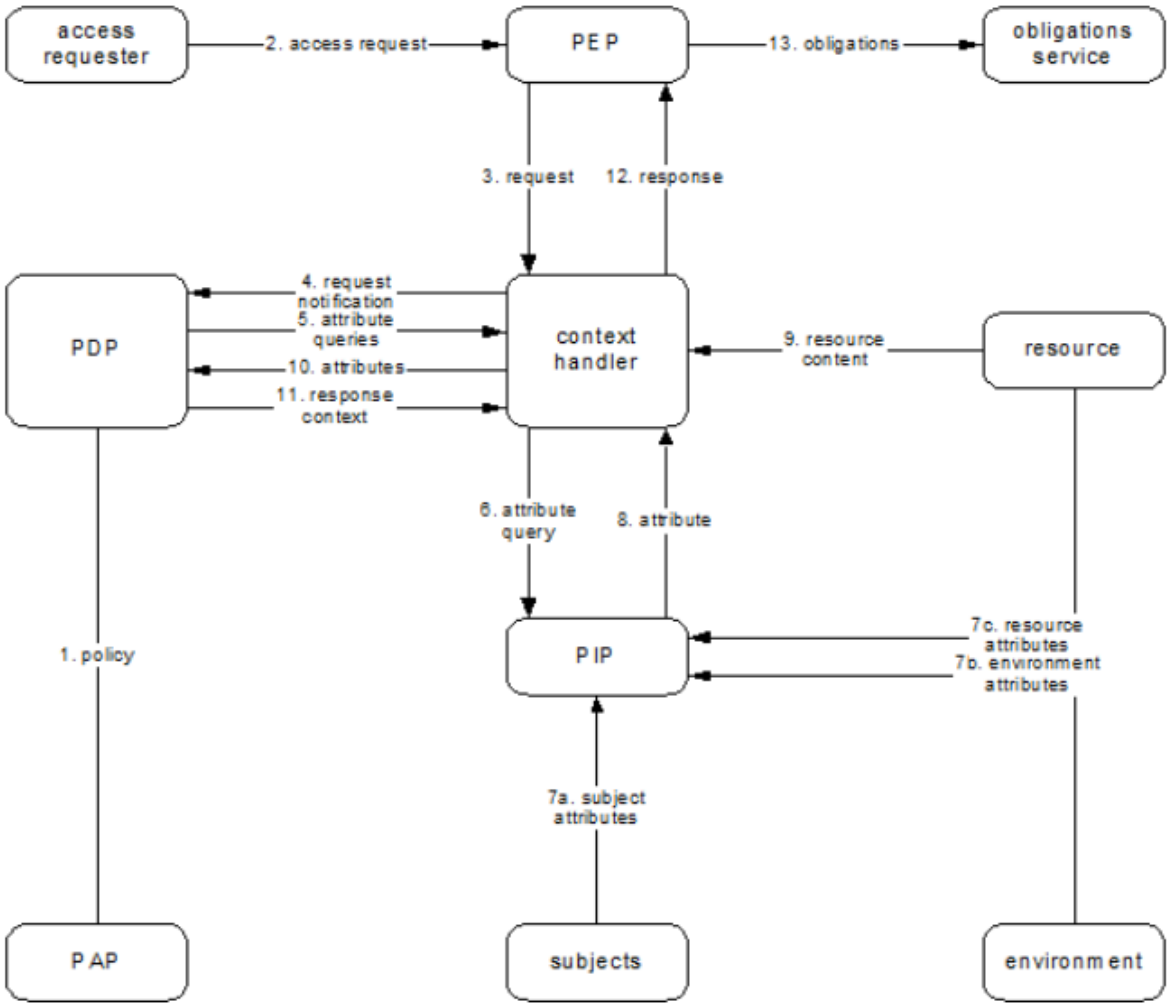


Figure 1.1: XACML data-flow diagram [2].

1.2.5 XACML 2.0 vs XACML 3.0

The two most widely used versions of XACML are XACML 2.0 and XACML 3.0. One key difference between the two, that plays a significant role in this paper, lies in the structure of their attributes. In XACML 2.0 attributes are organized into subject, resource, environment, and action categories using XML element tags. Whereas XACML 3.0 allows for definition of custom arbitrary attribute categories that can be indicated using XML attributes. The higher expressiveness of XACML 3.0 together with its increasing popularity is the reason that this

thesis will revolve around XACML 3.0 policies. Figure 1.2 shows an abstract syntax of XACML 3.0.

XACML Policy Components

```
<PolicySet> :- PolicySetID = [<Target>,≪ PolicySetID* ≫, CombID ]
             | PolicySetID = [<Target>, ≪ PolicyID* ≫, CombID ]
<Policy>    :- PolicyID = [<Target>, ≪ PolicySetID+ ≫ CombID ]
<Rule>      :- RuleID = [ Effect, <Target> , <Condition> ]
<Condition> :- propositional formulae
<Target>    :- Null
             | ⋀ <AnyOf> +
<AnyOf>     :- ⋁ <AllOf> +
<AllOf>     :- ⋀ <Match> +
<Match>     :- AttrType( attribute value )
CombID      :- po | do | fa | ooa
Effect      :- deny | permit
AttrType    :- subject | action | resource | environment
```

XACML Request Component

```
<Request>   :- { Attribute+}
Attribute   :- AttrType( attribute value ) | error(AttrType(attribute value))
             | external state
```

Figure 1.2: An abstract syntax of XACML 3.0 [1].

1.2.6 XACML 3.0 abstract syntax

1.2.6.1 Introduction

In a XACML policy, there are three primary elements: Rule, Policy, and Policy Set. The first primary element is the Rule, which has three components: an Effect, a Target, and a Condition. The effect is the evaluation of the rule and can take the values Permit, Deny, or Indeterminate [1]. The applicability of a rule to a set of access requests is specified by its Target which is made up of a set of AnyOf components in a conjunctive sequence. The AnyOf element is made up of a set of AllOf components in a disjunctive sequence. The AllOf element is made up of a set of Match component in a conjunctive sequence. The Match element uses a matching function to compare its attribute value to a request context's attribute value. The final component of the rule, the Condition, is a Boolean expression that refines the rule's applicability beyond its Target. The rule's Effect is returned if a request meets both the Targets and the Condition's requirements, otherwise, indeterminate is returned. The second primary

element is a policy, which has three components: a Target, a Rule combining algorithm, and a set of Rules. The Policy combines the Effects of its rules by using the Rule combining algorithm. The combining algorithm's result is returned if a request meets the Policy's Target's requirements, otherwise, indeterminate is returned. The third primary element, the Policy Set, is like the Policy, it is made up of a Target, a set of Policies and a Policy combining algorithm. Both the Rule combining algorithm and the Policy combining algorithm describe how the results of evaluating individual Rules inside a Policy or Policies inside a Policy Set respectively are combined to assess the Policy decision. The four Combining algorithms used in this syntax are first-applicable, permit-overrides, deny-overrides and only-one-applicable. Finally, the XACML Request component is a set of attribute types that are each assigned a value relative to their domain.

1.2.6.2 Target element

The Target element identifies the set of decision requests that the parent element is intended to evaluate. The Target elements appear as a component of a PolicySet and Policy element and can appear as a component of a Rule element. The target contains a conjunctive sequence of AnyOf components. The Target element may be absent from a Rule (empty Target). In this case, the target of the Rule is the same as that of the parent Policy element. An empty target matches any request. Otherwise, the target value will be "Match" if all the AnyOf specified in the target match values in the request context. Otherwise, if any one of the AnyOf specified in the target is "No Match", then the target will be "No Match". Otherwise, the target will be "Indeterminate". The target truth table is as follows:

<AnyOf> values	***Target*** value
All "Match"	"Match"
At least one "No Match"	"No Match"
Otherwise	"Indeterminate"

Table 1.1: Evaluation truth table of the Target element.

Throughout this thesis we use as our running example a real-world XACML 3.0 policy called "kmarket-gold-policy.xml". The full policy is listed in Appendix A.

```
<Target>
    <AnyOf>
        <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">gold
                </AttributeValue>
                <AttributeDesignator AttributeId="http://kmarket.com/id/role"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
            </Match>
        </AllOf>
    </AnyOf>
</Target>
```

Lines **7-19** of this example, specify the target of the policy. The one and only AnyOf element of the Target is matched when the value assigned to the "http://kmarket.com/id/role" attribute is “gold”. Overall, the target is matched if the value assigned to the "http://kmarket.com/id/role" attribute is “gold”.

```
<Target>
    <AnyOf>
        <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Liquor</AttributeValue>
                <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
            </Match>
        </AllOf>
    </AnyOf>
</Target>
```

Lines **41-52** specify the target of the second rule. The one and only AnyOf element of the Target is matched when the value assigned to the "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
attribute is “Liquor”. Overall, the target is matched if the value assigned to the "urn:oasis:names:tc:xacml:1.0:resource:resource-id" attribute is “Liquor”.

1.2.6.3 AnyOf element

The AnyOf element contains a disjunctive sequence of AllOf elements. The AnyOf element is matched by the values in the request context if at least one of its AllOf elements matches a value in the request context. The AnyOf truth table is as follows:

<AllOf> values	<AnyOf> Value
At least one "Match"	"Match"
None matches and at least one "Indeterminate"	"Indeterminate"
All "No match"	"No match"

Table 1.2: Evaluation truth table of the AnyOf element.

```
<AnyOf>
    <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">gold
            </AttributeValue>
            <AttributeDesignator AttributeId="http://kmarket.com/id/role"
            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </Match>
    </AllOf>
</AnyOf>
```

Lines **8-18** specify the one and only AnyOf of the target of the policy. The one and only AllOf element of the AnyOf is matched when the value assigned to the "http://kmarket.com/id/role" attribute is "gold". Overall, the AnyOf is matched if the value assigned to the "http://kmarket.com/id/role" attribute is "gold".

```
<AnyOf>
    <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Liquor</AttributeValue>
            <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </Match>
    </AllOf>
</AnyOf>
```

Lines **42-51** specify the one and only AnyOf of the target of the second rule. The one and only AllOf element of the AnyOf is matched when the value assigned to the "urn:oasis:names:tc:xacml:1.0:resource:resource-id" attribute is "Liquor". Overall, the AnyOf is matched if the value assigned to the "urn:oasis:names:tc:xacml:1.0:resource:resource-id" attribute is "Liquor".

1.2.6.4 AllOf element

The AllOf element contains a conjunctive sequence of Match elements. An AllOf is matched by the values in the request context if the value of all its Match elements is “True”. The AllOf truth table is as follows:

<Match> values	<AllOf> Value
All “True”	“Match”
No “False” and at least one “Indeterminate”	“Indeterminate”
At least one “False”	“No match”

Table 1.3: Evaluation truth table of the AllOf element.

```
<AllOf>
   <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">gold
      </AttributeValue>
      <AttributeDesignator AttributeId="http://kmarket.com/id/role"
      Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
      DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
   </Match>
</AllOf>
```

Lines **9-17** specify the one and only AllOf of the policy. The one and only Match element of the AllOf is matched when the value assigned to the "http://kmarket.com/id/role" attribute is “gold”. Overall, the AllOf is matched if the value assigned to the "http://kmarket.com/id/role" attribute is “gold”.

```
<AllOf>
   <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Liquor</AttributeValue>
      <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
      DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
   </Match>
</AllOf>
```

Lines **43-50** specify the one and only AllOf of the second rule. The one and only Match element of the AllOf is matched when the value assigned to the "urn:oasis:names:tc:xacml:1.0:resource:resource-id" attribute is “Liquor”. Overall, the AllOf is matched if the value assigned to the "urn:oasis:names:tc:xacml:1.0:resource:resource-id" attribute is “Liquor”.

1.2.6.5 Match element

The Match element identifies a set of entities by matching the attribute value of the request context with the embedded attribute value of the same attribute type. A Match is “True” if the embedded attribute value matches the attribute value in the request context.

```
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">gold
  </AttributeValue>
  <AttributeDesignator AttributeId="http://kmarket.com/id/role"
  Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
  DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
</Match>
```

Lines **10-16** specify the one and only Match of the policy. The embedded value of the attribute “http://kmarket.com/id/role" is “gold”. Overall, the Match is “True” if the value assigned in the request context to the "http://kmarket.com/id/role" attribute is “gold”.

```
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Liquor</AttributeValue>
  <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
  DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
</Match>
```

Lines **44-49** specify the one and only Match of the second rule. The embedded value of the attribute type "urn:oasis:names:tc:xacml:1.0:resource:resource-id” is “Liquor”. Overall, the Match is “True” if the value assigned in the request context to the "urn:oasis:names:tc:xacml:1.0:resource:resource-id" attribute is “Liquor”.

1.2.6.6 Effect element

The effect of the rule identifies the intended consequence of a "True" evaluation for the rule. Two values are allowed: "Permit" and "Deny".

```
<Rule Effect="Deny" RuleId="total-amount">
```

Line **20** specifies that the Effect of first rule of the policy is “Deny”.

```
<Rule Effect="Deny" RuleId="max-liquor-amount">
```

Line **40** specifies that the Effect of second rule of the policy is “Deny”.

```
<Rule RuleId="permit-rule" Effect="Permit"/>
```

Line **72** specifies that the Effect of third rule of the policy is “Permit”.

1.2.6.7 Condition element

Condition represents a Boolean expression that refines the applicability of the rule beyond the predicates implied by its target. The condition uses a function to compare its embedded attribute values to attribute values in the request context. It may be absent (empty), which means that it is satisfied by all requests. The condition value is "True" if the function returns “True”, otherwise the condition value is “False”.

```
<Condition>
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
         <AttributeDesignator AttributeId="http://kmarket.com/id/totalAmount"
         Category="http://kmarket.com/category"
         DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">1000</AttributeValue>
   </Apply>
</Condition>
```

Lines **21-30** specify the Condition of the first Rule of the policy. The embedded value of the attribute "http://kmarket.com/id/totalAmount" is 1000. The function of the condition is "urn:oasis:names:tc:xacml:1.0:function:integer-greater-than". Overall, the Condition is “True” if the value assigned in the request context to the "http://kmarket.com/id/totalAmount” attribute is greater than 1000.

```
<Condition>
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
         <AttributeDesignator AttributeId="http://kmarket.com/id/amount"
         Category="http://kmarket.com/category"
         DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
   </Apply>
</Condition>
```

Lines **53-62** specify the Condition of the second Rule of the policy. The embedded value of the attribute "http://kmarket.com/id/amount" is 10. The function of the condition is

"urn:oasis:names:tc:xacml:1.0:function:integer-greater-than". Overall, the Condition is “True” if the value assigned in the request context to the "http://kmarket.com/id/amount” attribute is greater than 10.

1.2.6.8 Rule element

The main components of a rule are the Target, the Condition, and the Effect. The Rule’s value can be calculated by evaluating its contents [2]. Rule evaluation involves separate evaluation of the Rule's Target and Condition. The Rule truth table is as follows:

Target	Condition	Rule evaluation
“Match” or no target	“True”	Effect
any	“False”	“Indeterminate”
“No-match”	any	“Indeterminate”

Table 1.4: Evaluation truth table of the Rule element.

```
<Rule Effect="Deny" RuleId="total-amount">
   <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
            <AttributeDesignator AttributeId="http://kmarket.com/id/totalAmount"
            Category="http://kmarket.com/category"
            DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true"/>
         </Apply>
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">1000</AttributeValue>
      </Apply>
   </Condition>
  <AdviceExpressions>
  <AdviceExpression AdviceId="deny-liquor-medicine-advice" AppliesTo="Deny">
  <AttributeAssignmentExpression AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">You are not allowed to do more
  than $1000 purchase from KMarket on-line trading system</AttributeValue>
  </AttributeAssignmentExpression>
  </AdviceExpression>
  </AdviceExpressions>
</Rule>
```

Lines **20-39** specify the first rule of the policy. This rule has an empty target indicated by the missing <Target> xml tag. The condition of the rule is satisfied when the total purchase amount from KMarket on-line trading system is above 1000$. This rule’s effect is Deny. Overall, the rule states that you are not allowed to do more than a $1000 total purchase from KMarket on-line trading system.

```
<Rule Effect="Deny" RuleId="max-liquor-amount">
<Target>
   <AnyOf>
      <AllOf>
         <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Liquor</AttributeValue>
            <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
         </Match>
      </AllOf>
   </AnyOf>
</Target>
   <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
            <AttributeDesignator AttributeId="http://kmarket.com/id/amount"
            Category="http://kmarket.com/category"
            DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true"/>
         </Apply>
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
      </Apply>
   </Condition>
 <AdviceExpressions>
 <AdviceExpression AdviceId="max-drink-amount-advice" AppliesTo="Deny">
 <AttributeAssignmentExpression AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text">
 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">You are not allowed to buy more
 tha 10 Liquor from KMarket on-line trading system</AttributeValue>
 </AttributeAssignmentExpression>
 </AdviceExpression>
 </AdviceExpressions>
</Rule>
```

Lines **40-71** specify the second rule of the policy. The target of the rule is satisfied when the resource trying to be accessed is Liquor. The condition of the rule is satisfied when the total number of bought items is above 10. This rule's effect is Deny. Overall, the rule states that you are not allowed to buy more than 10 Liquor from KMarket on-line trading system.

```
<Rule RuleId="permit-rule" Effect="Permit"/>
```

Line **72** specifies the third rule of the policy. This rule has an empty target indicated by the missing <Target> xml tag. This rule has an empty condition indicated by the missing <Condition> xml tag. This rule's effect is Permit. Overall, the rule is always evaluated to permit.

1.2.6.9 Policy element

The main components of a Policy are the Target, the Rule, and the Rule-combining algorithm [2]. The value of a Policy is determined by its contents, considered in relation to the contents of the request context. A policy's value is be determined by the evaluation of the Policy's Target and, according to the specified Rule-combining algorithm, which uses the values of the Policy's Rules. The Policy truth table is as follows:

Target	Rule-combining algorithm (based on rules)	Policy evaluation
"Match"	Decision	Decision
"No-match"	any	"Indeterminate"
"Indeterminate"	any	"Indeterminate"

Table 1.5: Evaluation truth table of the Policy element.

Lines **6-73** specify the one and only Policy of the Policy Set. The target of the Policy is satisfied when the value assigned to the "http://kmarket.com/id/role" attribute is “gold”. The Rule-Combining algorithm of the Policy is deny-overrides indicated by RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides". The Policy contains three rules indicated by three pairs of xml tags (<Rule…>, </Rule>) and are described in Section 1.2.6.2.

1.2.6.10 Policy Set element

The main components of a Policy Set are the Target, the Policy, the Policy Set, and the Policy-combining algorithm. The value of a Policy Set is determined by its contents, considered in relation to the contents of the request context. A Policy Set’s value is be determined by the evaluation of the Policy Set’s Target and, according to the specified Policy-combining algorithm, which uses the values of the Policy Set’s child Policies and child Policy Sets. The Policy Set truth table is as follows:

Target	Policy-combining algorithm	Policy Set evaluation
“Match”	Decision	Decision
“No-match”	any	“Indeterminate”
“Indeterminate”	any	“Indeterminate”

Table 1.6: Evaluation truth table of the Policy Set element.

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable"
    PolicySetId="RPSlist" Version="1.0">
    <Target />
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicyId="KmarketGoldPolicy" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides" Version="1.0">
   <Target>
      <AnyOf>
         <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">gold
               </AttributeValue>
               <AttributeDesignator AttributeId="http://kmarket.com/id/role"
               Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
               DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
            </Match>
         </AllOf>
      </AnyOf>
   </Target>
   <Rule Effect="Deny" RuleId="total-amount">
      <Condition>
         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
               <AttributeDesignator AttributeId="http://kmarket.com/id/totalAmount"
               Category="http://kmarket.com/category"
               DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">1000</AttributeValue>
         </Apply>
      </Condition>
    <AdviceExpressions>
    <AdviceExpression AdviceId="deny-liquor-medicine-advice" AppliesTo="Deny">
    <AttributeAssignmentExpression AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">You are not allowed to do more
    than $1000 purchase from KMarket on-line trading system</AttributeValue>
    </AttributeAssignmentExpression>
    </AdviceExpression>
    </AdviceExpressions>
   </Rule>

   <Rule Effect="Deny" RuleId="max-liquor-amount">
   <Target>
      <AnyOf>
         <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Liquor</AttributeValue>
               <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
               Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
               DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
            </Match>
         </AllOf>
      </AnyOf>
   </Target>
      <Condition>
         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
               <AttributeDesignator AttributeId="http://kmarket.com/id/amount"
               Category="http://kmarket.com/category"
               DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
         </Apply>
      </Condition>
    <AdviceExpressions>
    <AdviceExpression AdviceId="max-drink-amount-advice" AppliesTo="Deny">
    <AttributeAssignmentExpression AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">You are not allowed to buy more
    tha 10 Liquor from KMarket on-line trading system</AttributeValue>
    </AttributeAssignmentExpression>
    </AdviceExpression>
    </AdviceExpressions>
   </Rule>
    <Rule RuleId="permit-rule" Effect="Permit"/>
</Policy>
</PolicySet>
```

Lines **2-74** specify the one and only Policy Set of the XACML 3.0 real world policy. This rule has an empty target indicated by the first </Target> xml tag. The Rule-Combining algorithm of the Policy is first-applicable indicated by "PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable". The Policy Set contains a single Policy indicated by the pair of xml tags (<Policy…>, </Policy>) which is described in Section 1.2.6.8.

1.2.6.11 Policy/Rule Combining algorithm element

The rule-combining algorithm specifies the procedure by which the results of evaluating the component rules are combined when evaluating the policy, i.e. the decision value placed in the response context by the PDP is the value of the policy, as defined by the rule-combining algorithm. The policy-combining algorithm specifies the procedure by which the results of evaluating the component policies are combined when evaluating the Policy Set, i.e., the decision value placed in the response context by the PDP is the result of evaluating the Policy Set, as defined by the policy-combining algorithm. The abstract syntax of XACML3.0 defines four combining algorithms [2]: first-applicable, permit-overrides, deny-overrides, only-one-applicable.

```
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicyId="KmarketGoldPolicy" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides" Version="1.0">
```

Line **3** specifies the algorithm that will be used to resolve the results of the various policies that may be in the Policy Set. In this example, the policy combining algorithm is first applicable, indicated by PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable".

```
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicyId="KmarketGoldPolicy" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides" Version="1.0">
```

Line **6** specifies the algorithm that will be used to resolve the results of the various rule that may be in the policy. In this example, the rule combining algorithm is deny overrides, indicated by RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides".

1.2.6.11.1 Deny-overrides

The deny overrides combining algorithm is intended for those 1cases where a deny decision should have priority over a permit decision. This algorithm's behaviour is as follows:

1. If any decision is "Deny", the result is "Deny".
2. Otherwise, if any decision is "Indeterminate", the result is "Indeterminate".
3. Otherwise, if any decision is "Permit", the result is "Permit".

The rule/policy combining algorithm defined here has the following identifier:
urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides
or
urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides

1.2.6.11.2 Permit-overrides

The permit overrides combining algorithm is intended for those cases where a permit decision should have priority over a deny decision. This algorithm's behaviour is as follows:

1. If any decision is "Permit", the result is "Permit".
2. Otherwise, if any decision is "Indeterminate", the result is "Indeterminate".
3. Otherwise, if any decision is "Deny", the result is "Deny".

The rule/policy combining algorithm defined here has the following identifier:
urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides
or
urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides

1.2.6.11.3 First applicable

Each rule is evaluated in the order in which it is listed in the policy. For a particular rule, if the target matches and the condition evaluates to "True", then the evaluation of the policy is halted, and the corresponding effect of the rule will be the result of the evaluation of the policy (i.e. "Permit" or "Deny"). For a particular rule selected in the evaluation, if the target evaluates to "False" or the condition evaluates to "False", then the next rule in the order will be evaluated. If no further rule in the order exists, then the policy will evaluate to "Indeterminate".

The rule/policy combining algorithm defined here has the following identifier:
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable
or
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable

1.2.6.11.4 Only-one-applicable

In the entire set of policies in the Policy Set, if no policy is considered applicable by virtue of its target, then the result of the policy-combination algorithm will be "Indeterminate". If more than one policy is considered applicable by virtue of its target, then the result of the policy-combination algorithm will be "Indeterminate". If only one policy is considered applicable by evaluation of its target, then the result of the policy-combining algorithm will be the result of evaluating the policy.

The policy combining algorithm defined here has the following identifier:
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable

1.2.6.12 Request element

The request context is a list of values assigned to attribute types. Suppose we have a hypothetical decision request that might be submitted to a PDP that executes the policy of our running example (Appendix A). In English, the access request that generates the decision request may be stated as follows:
A requester with the role “gold” wants to buy 20 items called Liquor from the KMarket on-line trading system, which comes to a total of 550$.

In XACML, the information in the decision request is formatted into a request context statement that looks like the one in Appendix B.

```
<Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute IncludeInResult="false"
        AttributeId="http://kmarket.com/id/role">
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">gold</AttributeValue>
    </Attribute>
</Attributes>
```

Lines **7-13** specify that the value of attribute type "http://kmarket.com/id/role" is “gold”.

```
<Attributes Category="http://kmarket.com/category">
    <Attribute IncludeInResult="false"
        AttributeId="http://kmarket.com/id/totalAmount">
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">550</AttributeValue>
    </Attribute>
</Attributes>
```

Lines **14-20** specify that the value of attribute type "http://kmarket.com/id/totalAmount" is “550”.

```
<Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
    <Attribute IncludeInResult="false"
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
        <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">Liquor</AttributeValue>
    </Attribute>
</Attributes>
```

Lines **21-27** specify that the value of attribute type "urn:oasis:names:tc:xacml:1.0:resource:resource-id" is “Liquor”.

```
<Attributes Category="http://kmarket.com/category">
    <Attribute IncludeInResult="false"
        AttributeId="http://kmarket.com/id/amount">
        <AttributeValue
        DataType="http://kmarket.com/id/amount">20</AttributeValue>
    </Attribute>
</Attributes>
```

Lines **28-34** specify that the value of attribute type "http://kmarket.com/id/amount" is “20”.

Response evaluation process is as follows:

The Condition of the first Rule (line 21-30 of Appendix A) is evaluated to “False” since 550>1000 is not true, where 550 is the attribute value assigned by the request context to the attribute type “http://kmarket.com/id/totalAmount” (line 14-20 of Appendix B), 1000 is the embedded attribute value of the Condition for the attribute type “http://kmarket.com/id/totalAmount” (line 28 of Appendix A), and > is the function used by the Condition (line 22 of Appendix A).

The Condition of the second Rule (line 53-62 of Appendix A) is evaluated to “True” since 20>10 is true, where 20 is the attribute value assigned by the request context to the attribute type "http://kmarket.com/id/amount” (line 28-34 of Appendix B), 10 is the embedded attribute value of the Condition for the attribute type "http://kmarket.com/id/amount” (line 60 of Appendix A), and > is the function used by the Condition (line 54 of Appendix A). The Condition of the third Rule (line 72 of Appendix A) is evaluated to “True” since it has not condition.

The Match of the target of the policy is evaluated to “True” since the embedded attribute value (line 11 of Appendix A) “gold” matches the attribute value assigned by the request context (line 7-13 of Appendix B) for the attribute type "http://kmarket.com/id/role" (line 13 of Appendix A, line 9 of Appendix B). Since Match is “True” and it is the only Match component of the AllOf (line 9-17 of Appendix A), that AllOf is matched. The AnyOf (line 8-18 of Appendix A) is matched since one of its AllOf components is matched. In turn the Target of the Policy is matched (line 7-19 of Appendix A) since it has a single AnyOf component, and it has been matched.

The Match of the target of the second rule is evaluated to “True” since the embedded attribute value (line 45 of Appendix A) “Liquor” matches the attribute value assigned by the request context (line 21-27 of Appendix B) for the attribute type "urn:oasis:names:tc:xacml:1.0:resource:resource-id" (line 46 of Appendix A, line 23 of Appendix B). Since Match is “True” and it is the only Match component of the AllOf (line 9-17 of Appendix A), that AllOf is matched. The AnyOf (line 42-51 of Appendix A) is matched since one of its AllOf components is matched. In turn the Target of the second Rule is matched (line 41-52 of Appendix A) since it has a single AnyOf component, and it has been matched.

The first Rule is evaluated “indeterminate” since its Condition was evaluated “False”. The second rule is evaluated “Deny” since its Effect is “Deny” (line 40 of Appendix A), its Target was matched and its Condition was evaluated “True”. The third Rule is evaluated “Permit” since its Effect is “Permit” (line 72 of Appendix A), its Target is empty, and it has no Condition.

The Policy is evaluated to “Deny”, since its Target was matched, and it uses the deny-override Rule combining algorithm” (line 6 of Appendix A) that comes to a decision “Deny” because the second Rule was evaluated to “Deny”. The Policy Set is evaluated to “Deny”, since it has an empty Target, and it uses the first-applicable Policy combining algorithm” (line 3 of Appendix A) that comes to a decision “Deny” because the one and only Policy was evaluated to “Deny”. Thus, the final response to the request is “Deny”.

1.3 Answer Set Programming (ASP)

1.3.1 Introduction to ASP

Answer Set Programming (ASP) is a declarative logic programming language that helps us solve search problems. Unlike traditional programming languages that use algorithms to solve their problems, a program in a declarative language simply describes what is counted as a solution. Using this description, the declarative programming system discovers a solution by the process of automated reasoning [6]. In other words, a declarative program encodes the problem itself, it is made up of a set of conditions on the values of variables that define solutions to the problem, also called answer sets or stable models.

1.3.1.1 ASP syntax

Any program written in a logic programming language is a set of sentences in logical form, expressing facts and rules about some problem domain. An ASP program is a logic program made up of a finite set of declarative rules of the form [5]:

$$a_0 :- a_1, ..., a_m, \text{not } a_m+1, ..., \text{not } a_n. \quad (1)$$

where $0 \leq m \leq n$ and $a_0, \ldots, a_n$ are propositional atoms. The atom a_0 is the head of the rule (1), and the list $a_1, ..., a_m, \text{not } a_m+1, ..., \text{not } a_n$ is its body. If the body is empty, then the rule is known as a fact. The rule above states that a_0 must be true if $a_1, ..., a_m$ have been proven true and if $a_m+1, ..., a_n$ could not be proven true, meaning that they are possibly false.

Rule (2) has a head which includes atoms in braces [6]. This is a "choice-rule" which describes all possible ways to choose which of the atoms p or q or r are included in the stable model. Suppose we have a program that consists of the following rule:

$$\{p, q, r\}. \quad (2)$$

Answer sets of this one-rule program are: ∅, {p}, {q}, {r}, {p, q}, {p, r}, {q, r}, {p, q, r}.

These rules can also have integers before or after the braces, which express bounds on the cardinality of the stable model described by the rule. Suppose we have a program that consists of the following rule:

$$1\{p, q, r\}2.$$

Answer sets of this one-rule program are: {p}, {q}, {r}, {p, q}, {p, r}, {q, r}.

Rule (4) has an empty head. This is a “constraint-rule” which eliminates all answer sets of a program where p has been proven and q could not be proven.

:- p, not q. (3)

1.3.1.2 Answer Sets

The definition of an answer set (stable model) tells us when a model of a propositional formula F (that is, a truth assignment satisfying F) is considered “stable” [12]. It provides a semantics for grounded ASP programs in view of two conventions. First, we agree to treat rules and programs without variables as propositional formulas “written in logic programming notation”. For instance, we identify the second convention is to identify any set X of atoms with the truth assignment that makes all elements of X true and makes all other atoms false. The reduct F_X of a propositional formula F relative to a set X of atoms is the formula obtained from F by replacing each maximal sub formula that is not satisfied by X with ⊥ (falsity). We say that X is a stable model of F if X is minimal among the sets satisfying F_X. The minimality of X is understood here in the sense of set inclusion: no proper subset of X satisfies F_X.
For example, consider the following rules of a logic program:

p(1). p(2). p(3).
q(3) :- notr(3).
r(X) :- p(X), not q(X).

The ground rules of the propositional program F are:

p(1). p(2). p(3).
q(3) :- not r(3).
r(1) :- p(1), not q(1).
r(2) :- p(2), not q(2).
r(3) :- p(3), not q(3).

Now, suppose we want to check whether X = {p(1), p(2), p(3)} is an answer set of the grounded program F shown above. The reduct F_X of the program with respect to X is as follows:

p(1). p(2). p(3). q(3).
r(1) :- p(1).
r(2) :- p(2).
r(3) :- p(3).

All the atoms we can prove from reduct F_X are {p(1), p(2), p(3), r(1), r(2), r(3), q(3)} which are different from X={p(1), p(2), p(3)}, thus X is not an answer set of this program. Now, suppose we want to check if X' = {p(1), p(2), p(3), q(3), r(1), r(2)} is an answer set. The reduct $F_{X'}$ of the program with respect to this new answer set is as follows:

p(1). p(2). p(3). q(3).
r(1) :- p(1).
r(2) :- p(2).

All the atoms we can prove from reduct $F_{X'}$ are {p(1), p(2), p(3), q(3), r(1), r(2)} which is identical to X'. This means that X'= {p(1), p(2), p(3), q(3), r(1), r(2)} is an answer set of this program.

1.3.2 ASP workflow

The ASP workflow is as follows. Initially we model the problem using first logic programming. Next, we proceed with grounding, a process that removes all the variables from our initial program without losing any of its solutions (answer sets/stable models). After producing the ground program (a propositional representation of the original ASP program), we use this representation to generate the answer sets (stable models), also known as solving [1]. The workflow is shown in Figure 3.

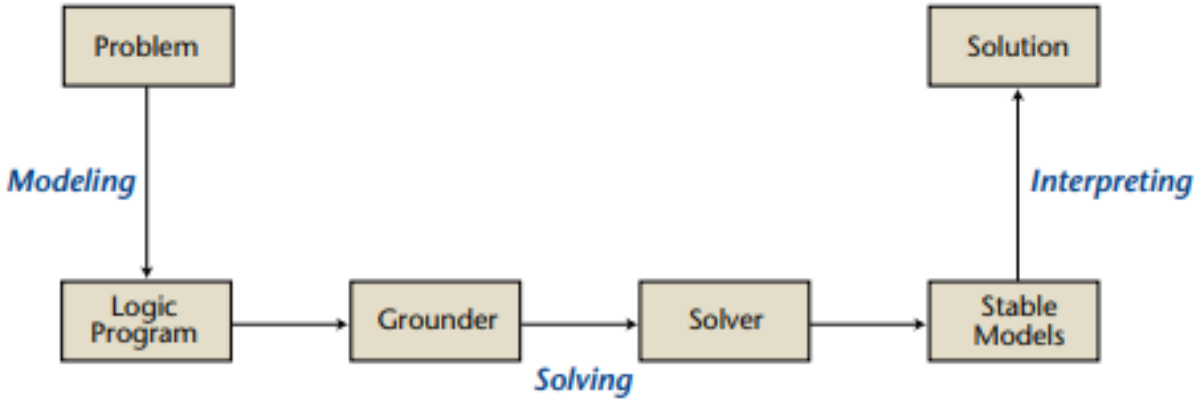


Figure 1.3: The workflow of ASP [3].

1.3.2.1 ASP example

Suppose we have a simple problem problem where we want to color vertexes of a graph with one of two colors. We model this problem into an ASP logic problem P with the following rules, choices, and facts:

vertex(0). (1)
vertex(1). (2)
col(red). (3)
col(blue). (4)
$1\{color(X,C) : col(C)\}1 \leftarrow vertex(X)$. (5)

There are 2 vertexes indicated by facts (1-2), and 2 colors indicated by facts (3-4), We state that each vertex must be assigned with exactly 1 color indicated by rule(5).

The grounder takes the above logic program P and produces a variable-free ground program G_P. The ground instantiation of the logic program P is the set of all the ground instances of rules of P that can be obtained by substituting variables with constants [11]. The ground program G_P contains the following:

vertex(0).
vertex(1).

col(red).

col(blue).

1{color(0,0) : col(0)}1 ← vertex(0).

1{color(1,1) : col(1)}1 ← vertex(1).

1{color(red,red) : col(red)}1 ← vertex(red).

1{color(blue,blue) : col(blue)}1 ← vertex(blue).

1{color(blue,red) : col(red)}1 ← vertex(blue).

1{color(blue,0) : col(0)}1 ← vertex(blue).

1{color(blue,1) : col(1)}1 ← vertex(blue).

1{color(red,blue) : col(blue)}1 ← vertex(red).

1{color(red,0) : col(0)}1 ← vertex(red).

1{color(red,1) : col(1)}1 ← vertex(red).

1{color(0,blue) : col(blue)}1 ← vertex(0).

1{color(0,red) : col(red)}1 ← vertex(0).

1{color(0,1) : col(1)}1 ← vertex(0).

1{color(1,0) : col(0)}1 ← vertex(1).

1{color(1,red) : col(red)}1 ← vertex(1).

1{color(1,blue) : col(blue)}1 ← vertex(1).

Some ground rules are useless. They will never be applicable because their bodies contain atoms that are not derivable from the program (they do not appear in the head of any rule). Such rules are ones that have atoms like vertex(red/blue), or col(0/1), or color(blue/red,..), or color(…,0/1) in them. Grounding, may be computationally very expensive having a big impact on the performance of the whole system.

The solver takes the grounded program G_P above and generates the following stable models (answer sets):

Answer1

{color(0, red), color(1, blue), col(blue), col(red),vertex(1), vertex(0)}

Interpreted as: color the first vertex red and the second vertex blue.

Answer2

{color(0, red), color(1, red), col(blue), col(red),vertex(1), vertex(0)}

Interpreted as: color the first vertex red and the second vertex red.

Answer3

{color(0, blue), color(1, blue), col(blue), col(red),vertex(1), vertex(0)}
Interpreted as: color the first vertex blue and the second vertex blue.
Answer4
{color(0, blue), color(1, red), col(blue), col(red),vertex(1), vertex(0)}
Interpreted as: color the first vertex blue and the second vertex red.

1.3.3 Clingo

Clingo is a tool used for grounding and solving logic programs, developed at the University of Potsdam. Clingo combines the functionalities of two other tools called gringo, and clasp [4]. The first tool called gringo is a grounder which takes the original program and compute an equivalent variable-free program. The second tool called clasp is a solver which takes the propositional programs generated by the grounder and computes the answer sets. Another useful functionality of Clingo is that it can integrate the scripting language Python which will later help us compare attribute type values.

Chapter 2

XACML to ASP translation and XACBench

2.1 Motivation 24

2.2 Proposed framework and prototype tool for translation and analysis of XACML policies 25

2.2.1 Framework for policy translation and analysis 25

2.2.2 The XACBench prototype tool for policy translation 26

2.3 ASP programs of the top framework layer 27

2.3.1 Mapping policy dependent rules 27

2.3.2 Mapping policy independent rules 30

2.3.3 Mapping combining algorithms 38

2.3.3.1 Permit overrides 38

2.3.3.2 Deny overrides 41

2.3.3.3 First applicable 43

2.3.3.4 Only one applicable 45

2.3.4 Mapping requests 46

2.4 ASP programs of the middle framework layer 47

2.4.1 Reachability 48

2.4.2 Usefulness 49

2.4.3 Total redundancy 50

2.4.4 Completeness 52

2.4.5 Isomorphism 52

2.4.6 Intra-policy anomalies 54

2.4.7 Inter-policy anomalies 56

2.5 ASP request generator 59

2.1 Motivation

Today, many organisations, governments, and companies have adopted the use of Web applications. Such applications hold valuable information of organisations that have adopted them, and their customers. The need for protection of digital information has led to the use of access control policies which allow applications that implement them to define rules and guidelines structuring who can access data and resources at an organization. One of the most common access control policy languages used today is the eXtensible Access Control Markup Language (XACML), which lets users define access policies made up of complicated conditions. Despite its popularity and its rich and expressive model for the specification of access policies, XACML does not have an effective way to analyse the policies that it specifies.

It is essential that the process of creating new policies, updating, or removing ones that already exist is followed by a thorough analysis of the newly defined set of policies. Updates to access policies may cause errors, and anomalies which can in turn lead to incorrect decision-making or even an application that is susceptible to attacks. Analysis is crucial for detecting the impact of different changes made to access policies, whether anomalies or conflicts have arisen as an aftereffect. Performing such an analysis manually is impractical and is even more daunting when the policies are defined by multiple authorities, with each authority having their own set of policies. Therefore, there is a need for a tool which will allow policy developers and analysers to examine a wide range of properties effectively and efficiently both at policy design time, and during the general maintenance phase.

In this chapter we will be going over the idea proposed by Mohsen Rezvani, David Rajaratnam, Aleksandar Ignjatovic, Maurice Pagnucco,and Sanjay Jha, which is to translate XACML policies into sets of ASP equivalent programs [1]. We will also be going over a prototype translation tool, that follows this idea called XACBench, developed by Shayan Ahmadi , Mohammad Nassiri , and Mohsen Rezvani [9].

2.2 Proposed framework and prototype tool for translation and analysis of XACML policies

2.2.1 Framework for policy translation and analysis

Here we present the framework for XACML policy analysis using ASP proposed by Mohsen Rezvani, David Rajaratnam, Aleksandar Ignjatovic, Maurice Pagnucco, and Sanjay Jha. This framework initially translates XACML policies together with a wide range of policy properties into equivalent ASP programs and then uses an ASP solver to generate answer sets which are then used to verify those policy properties [1]. The top framework layer maps input XACML policies to an equivalent set of ASP programs. This layer is divided into two parts. The first part are policy independent ASP programs; it contains the ASP translations of XACML combining algorithms and the match programs which are common for all input XACML policies. The second part is the policy dependent ASP program; an ASP program that contains facts that describe the input XACML policy. These facts specified by the input XACML policy describe the components of the policy, who they belong to, and under which circumstances they can be satisfied or matched (Section 2.3.4). In the middle framework layer, there are policy independent property programs with ASP rules that help verify a wide range of XACML policy properties (redundancies, completeness, generalizations, etc.). ASP programs of the top and middle framework layer are passed together to an ASP solver, such as Clingo. The answer sets generated by the solver are then used to verify properties that stand true for the input XACML policy. Policy analysis can be divided into two main processes. The first process is query analysis, where we check what authorization decision the XACML policy comes to when faced with some request. The other analysis process is policy property analysis, where we check policy properties such as total redundancy, simple redundancy, shadow anomaly, correlation anomaly, generalization anomaly, reachability, usefulness, isomorphism, and completeness. The policy analysis framework is shown in Figure 2.1.

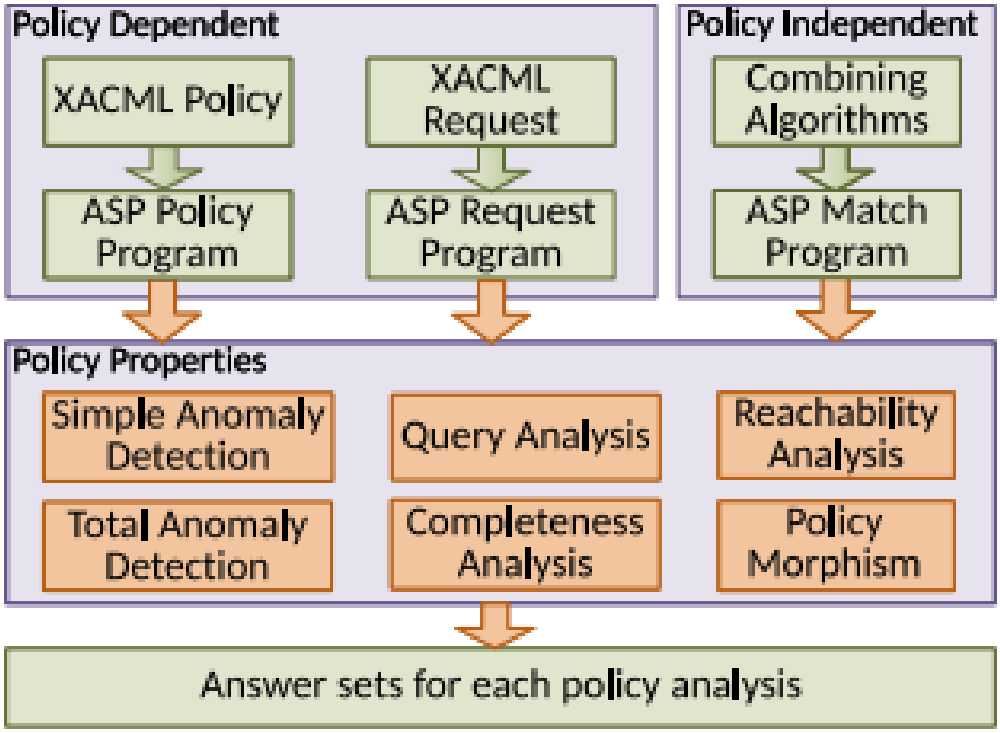


Figure 2.1: Policy analysis framework [1].

2.2.2 The XACBench prototype tool for policy translation

Here we present the prototype tool called XACBench developed by Shayan Ahmadi, Mohammad Nassiri, and Mohsen Rezvani. XACBench is a typical Maven Java project which offers two main functionalities that are used in this thesis [9]. The first and essential functionality is the translation of XACML 3.0 policies to ASP equivalent programs that follows the framework described above. The second functionality lets us generate synthetic XACML security policies of varying sizes, which is used in this thesis to help us run tests on policies of varying sizes and complexity.

In the XACML 3.0 to ASP translation functionality, XACBench takes a XACML 3.0 policy as input and converts it into a set of ASP equivalent programs, the policy dependent programs. As for the input policy independent ASP programs, XACBench contains a predefined set of ASP programs that describe combining algorithms, matching, and a wide range of policy properties.

2.3 ASP programs of the top framework layer

To analyse various properties of an XACML policy we need ASP programs equivalent to the XACML policy and its components. The XACBench tool contains ASP programs such as “match_target_v3.asp”, and “match_common.asp” which contain the policy independent ASP matching programs, Sections 2.3.2-2.3.3 exhibit rules within these programs. XACBench also generates an ASP program that contains the policy dependent ASP program, Section 2.3.4 exhibits rules within this program. In other words, this program contains facts written using the ASP language that are equivalent to the facts that described the input access policy in the XACML language.

2.3.1 Mapping policy dependent rules

In Figure 2.2 we can see the result of using the XACBench XACML 3.0 to ASP translation functionality on our running example, the real-world XACML 3.0 policy called “kmarket-gold-policy.xml”.

```
6   target(r1).
7   condition(r1, true).
8   rule(r1, 1, p1, deny).
9   anyof(r2, any2, any, liquor).
10  target(r2) :- anyof(r2 ,any2, _, _).
11  condition(r2, true).
12  rule(r2, 2, p1, deny).
13  target(r3).
14  condition(r3, true).
15  rule(r3, 3, p1, permit).
16  anyof(p1, any2, gold , any).
17  target(p1) :- anyof(p1 ,any2, _, _).
18  policy(p1, 1, ps0, deny_overrides).
19  target(ps0).
20  policy_set(ps0, 0, ps0, first_applicable).
```

Figure 2.2: ASP equivalent to the XACML 3.0 policy called kmarket-gold-policy.

(1)
anyof(rid, anyid, action_value, environment_value, resource_value, subject_value).

Rule (1) is a fact which states that there is an AllOf element that belongs an AnyOf element anyid of Rule rid with embedded attribute values action_value, environment_value, resource_value, and subject_value.

We can see this fact rule appear in lines 9, 16 of Figure 2.2. These facts are generated based on lines 7-19, and 41-52 of Appendix A.

(2)
target(rname).

Rule (2) is a fact which states rule/policy/Policy Set rname has an empty target.
We can see this fact rule appear in lines 6, 13, 19 of Figure 2.2. Line 19 states that the policy set has an empty target and is based on line 5 of Appendix A. Line 13 states that the third Rule of the Policy has an empty target based on the absence of the <Target> xml tag in line 72 of Appendix A. Line 6 states that the first Rule of the Policy has an empty target based on the absence of the <Target> xml tag in lines 20-39 of Appendix A.

(3)
target(rname) :- anyof(rname ,anyid, _, _).

Rule (3) states that a rule/policy/policy set has an empty target if it also has an AnyOf element. This rule causes issues in our translation which we discuss in Section 3.4. We can see this fact rule appear in lines 10, 17 of Figure 2.2 since both the Policy and its second Rule have Targets indicated by lines 7-19 and 41-52 of Appendix A respectively. We discuss how this part must be changed in the next chapter.

(4)
condition(rname, true).

Rule (4) is a fact which states that rule rname has an empty condition. We can see this fact rule appear in lines 7, 11, 14 of Figure 2.2. This part of the translation is incorrect and we discuss it in the next Chapter.

(5)
condition(rname, is_subject(subject_value)).

Rule (5) is a fact which states that rule rname has a condition that the access-subject is subject_value

(6)
condition(rname, is_subject_resource(subject_value, resource_value)).

Rule (6) is a fact which states that rule rname has a condition that the access-subject is subject_value and the access-resource is resource_value

(7)
rule(rname, rid, pname, ef).

Rule (7) is a fact which states that rname is a rule with id rid and effect ef that belongs to policy pname. We can see this fact rule appear in lines 8, 12, 15 of Figure 2.2. Line 8 states that there is a Rule r1 of Policy p1 with the Effect deny, which is based on lines 20-39 of Appendix A. Line 12 states that there is a Rule r2 of Policy p1 with the Effect deny, which is based on lines 40-71 of Appendix A. Line 15 states that there is a Rule r3 of Policy p1 with the Effect permit, which is based on line 72 of Appendix A.

(8)
policy(pname, pid, psname, alg).

Rule (8) is a fact which states that pname is a policy with id pid, that uses the combining algorithm alg, and belongs to Policy Set psname. We can see this fact rule appear in line 18 of Figure 2.2. Line 18 states that there is a Policy p1 that belongs to Policy Set ps0 that use the combining algorithm deny-overrides, which is based on lines 6-73 of Appendix A.

(9)
policy_set(psname, psid, ppsname, alg).

Rule (9) is a fact which states that psname is a Policy Set with id psid, that uses the combining algorithm alg, and belongs to Policy Set ppsname. If the value of psid is 0, then it is the root Policy Set and ppsname is equal to psname. We can see this fact rule appear in line 20 of Figure

2.2. Line 20 states that there is a Policy Set ps0 that uses the combining algorithm first-applicable, which is based on lines 2-74 of Appendix A.

2.3.2 Mapping policy independent rules

A target in XACML 3.0 matches a request if all AnyOf objects in the target match the request. An AnyOf object matches a request if at least one of its AllOf objects matches the request.

(10)

```
match_anyof_req(RN, AnyID, ActReq, EnvReq, ResReq, SubReq) :-
                                        request(ActReq, EnvReq, ResReq, SubReq),
                                        anyof(RN, AnyID, ActAnyof, EnvAnyof, ResAnyof,
                                        SubAnyof),
                                        @match_str_func(ActReq, ActAnyof, any) == 1,
                                        @match_str_func(EnvReq, EnvAnyof, any) == 1,
                                        @match_str_func(ResReq, ResAnyof, any) == 1,
                                        @match_str_func(SubReq, SubAnyof, any) == 1.
```

Rule (10) states that an AllOf element of rule/policy RN with id AnyID is matched to a request with values ActReq, EnvReq, ResReq, SubReq under certain conditions.
The first condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The second condition is that there is an AllOf element that belongs to rule/policy RN with id AnyID with ActAnyof, EnvAnyof, ResAnyof, SubAnyof as values assigned to the attributes of the policy, indicated by the atom [anyof(RN, AnyID, ActAnyof, EnvAnyof, ResAnyof, SubAnyof)]. The last four conditions use a python script

```
#script (python)
def match_str_func(str1, str2, any_str):
   if (str1 == str2) or (str2 == any_str):
      return 1
   else:
      return 0
#end.
```

to check whether the values (ActAnyof, EnvAnyof, ResAnyof, SubAnyof) assigned to the attributes by the AllOf element match those of the request (ActReq, EnvReq, ResReq, SubReq) respectively. If a value assigned to an attribute by the AllOf element is any, then the condition for that attribute is satisfied for all values of the request for that same attribute.

(11)

```
no_match_anyof_req(RN, ActReq, EnvReq, ResReq, SubReq) :-
                                    request(ActReq, EnvReq, ResReq, SubReq),
                                    anyof(RN, AnyID, _, _, _, _),
                                    not match_anyof_req(RN, AnyID, ActReq, EnvReq,
                                    ResReq, SubReq).
```

Rule (11) states that an AnyOf element of rule/policy RN is not matched to a request with values ActReq, EnvReq, ResReq, SubReq under certain conditions.
The first condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The second condition is that there is an AllOf element that belongs to rule/policy RN with id AnyID, indicated by the atom [anyof(RN, AnyID, _, _, _, _)]. The third condition is that an AllOf element with id AnyID, that belongs to the rule/policy RN with ActReq, EnvReq, ResReq, SubReq as values assigned to the attributes of the policy could not be proven, indicated by the atom [not match_anyof_req(RN, AnyID, ActReq, EnvReq, ResReq, SubReq)].

(12)

```
match_target(TN, ActReq, EnvReq, ResReq, SubReq) :-
                                    request(ActReq, EnvReq, ResReq, SubReq),
                                    target(TN).
```

Rule (12) states that a target element of rule/policy TN is matched to a request with values ActReq, EnvReq, ResReq, SubReq under certain conditions.
The first condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The second condition is that the rule/policy TN has an empty target, indicated by the atom [target(TN)].

(13)

```
match_target(TN, ActReq, EnvReq, ResReq, SubReq) :-
                                    request(ActReq, EnvReq, ResReq, SubReq),
                                    anyof(TN, _, _, _, _, _),
                                    not no_match_anyof_req(TN, ActReq, EnvReq, ResReq,
                                    SubReq).
```

Rule (13) states that a target element of rule/policy TN is matched to a request with values ActReq, EnvReq, ResReq, SubReq under certain conditions.
The first condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The second condition is that the rule/policy TN has an AllOf element, indicated by the atom [anyof(TN, _, _, _, _, _)]. The third condition is that there isn't a single AnyOf element of rule/policy TN that could not be matched to a request with values ActReq, EnvReq, ResReq, SubReq, indicated by the atom [not no_match_anyof_req(TN, ActReq, EnvReq, ResReq, SubReq)].

(14)
bool_expr(true).

Rule (14) seems to have no use in the general scheme since it does not appear in any body of any rule defined in XACBench.

(15)
```
bool_expr(true, ActReq, EnvReq, ResReq, SubReq) :-
                                        request(ActReq, EnvReq, ResReq, SubReq).
```

Rule (15) states that the Boolean expression true is satisfied by a request with values ActReq, EnvReq, ResReq, SubReq under certain conditions.
The first condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)].

(16)
```
bool_expr(is_subject(X), ActReq, EnvReq, ResReq, SubReq) :-
                                        condition(_, is_subject(X)),
                                        request(ActReq, EnvReq, ResReq, SubReq),
                                        X == SubReq.
```

Rule (16) states that the Boolean expression is_subject(X) is satisfied by a request with values ActReq, EnvReq, ResReq, SubReq under certain conditions.
The first condition is that there is some rule with a condition that is satisfied only if the value of the subject attribute is X, indicated by the atom [condition(_, is_subject(X))]. The second

condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that X must be equal to the request's value SubReq assigned to the subject attribute, indicated by the atom [X == SubReq].

(17)

```
bool_expr(is_subject_resource(X,Y), ActReq, EnvReq, ResReq, SubReq) :-
                                        condition(_, is_subject_resource(X, Y)),
                                        request(ActReq, EnvReq, ResReq, SubReq),
                                        X == SubReq,
                                        Y == ResReq.
```

Rule (17) states that the Boolean expression is_subject_resource(X,Y) is satisfied by a request with values ActReq, EnvReq, ResReq, SubReq under certain conditions.
The first condition is that there is some rule with a condition that is satisfied only if the value of the subject attribute is X and the value of the resource attribute is Y, indicated by the atom [condition(_, is_subject_resource(X, Y))]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that X must be equal to the request's value SubReq assigned to the subject attribute, indicated by the atom [X == SubReq]. The fourth condition is that X is equal to the request's value SubReq assigned to the subject attribute, indicated by the atom [X == SubReq]. The fifth condition is that Y must be equal to the request's value ResReq assigned to the resource attribute, indicated by the atom [Y == ResReq].

(18)

```
match_rule(RN, RID, PN, E, ActReq, EnvReq, ResReq, SubReq) :-
                                        rule(RN, RID, PN, E),
                                        request(ActReq, EnvReq, ResReq, SubReq),
                                        match_target(RN, ActReq, EnvReq, ResReq, SubReq),
                                        condition(RN, B),
                                        bool_expr(B, ActReq, EnvReq, ResReq, SubReq).
```

Rule (18) states that a rule RN with id RID that belongs to the policy PN will have the effect E on the request with values ActReq, EnvReq, ResReq, SubReq under certain conditions.

The first condition is that there is a rule RN with id RID, and effect E, that belongs to policy PN, indicated by the atom [rule(RN, RID, PN, E)]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that the target of rule RN is matched to a request with values ActReq, EnvReq, ResReq, SubReq, indicated by the atom [match_target(RN, ActReq, EnvReq, ResReq, SubReq)]. The fourth condition is that the rule RN has a condition with a boolean expression B, indicated by the atom [condition(RN, B)].
The fifth condition is that the Boolean expression B is satisfied by a request with values ActReq, EnvReq, ResReq, SubReq, indicated by the atom [bool_expr(B, ActReq, EnvReq, ResReq, SubReq)].

(19)

```
match_policy(PN, PID, PS, RN, E, ActReq, EnvReq, ResReq, SubReq) :-
                                    policy(PN, PID, PS, ALG),
                                    request(ActReq, EnvReq, ResReq, SubReq),
                                    match_target(PN, ActReq, EnvReq, ResReq, SubReq),
                                    match_policy_alg(PN, RN, ALG, E, ActReq, EnvReq,
                                    ResReq, SubReq).
```

Rule (19) states that a policy PN with id PID that belongs to the Policy Set PS will have the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN under certain conditions.
The first condition is that there is a policy PN with id PID, that belongs to policy PS and uses the combining algorithm ALG, indicated by the atom [policy(PN, PID, PS, ALG)]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that the target of policy PN is matched to a request with values ActReq, EnvReq, ResReq, SubReq, indicated by the atom [match_target(PN, ActReq, EnvReq, ResReq, SubReq)]. The fourth condition is that the policy PN using the combining algorithm ALG will have the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN, indicated by the atom [match_policy_alg(PN, RN, ALG, E, ActReq, EnvReq, ResReq, SubReq)].

(20)

```
match_policyset(PSN, PSID, PPS, CN, RN, E, ActReq, EnvReq, ResReq, SubReq) :-
                                    policy_set(PSN, PSID, PPS, ALG),
```

request(ActReq, EnvReq, ResReq, SubReq),
match_target(PSN, ActReq, EnvReq, ResReq, SubReq),
match_policyset_child(CN, _, PSN, RN, E, ActReq, EnvReq, ResReq, SubReq),
match_policyset_alg(PSN, CN, RN, ALG, E, ActReq, EnvReq, ResReq, SubReq).

Rule (20) states that a Policy Set PSN with id PSID that belongs to the Policy Set PPS will have the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN that belongs to child policy CN under certain conditions.
The first condition is that there is a Policy Set PSN with id PSID, that belongs to Policy Set PPS and uses the combining algorithm ALG, indicated by the atom [policy(PSN, PSID, PPS, ALG)]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that the target of policy PSN is matched to a request with values ActReq, EnvReq, ResReq, SubReq, indicated by the atom [match_target(PN, ActReq, EnvReq, ResReq, SubReq)]. The fourth condition is that a child policy CN of the Policy Set PSN has the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN, indicated by the atom [match_policyset_child(CN, _, PSN, RN, E, ActReq, EnvReq, ResReq, SubReq)]. The fifth condition is that the Policy Set PSN using the combining algorithm ALG has the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN that belongs to child policy CN, indicated by the atom [match_policyset_alg(PSN, CN, RN, ALG, E, ActReq, EnvReq, ResReq, SubReq)].

(21)

match_policyset(PSN, PSID, PPS, 0, 0, indeterminate, ActReq, EnvReq, ResReq, SubReq) :-
policy_set(PSN, PSID, PPS, ALG),
request(ActReq, EnvReq, ResReq, SubReq),
not match_policyset(PSN, PSID, PPS, _, _, permit, ActReq, EnvReq, ResReq, SubReq),
not match_policyset(PSN, PSID, PPS, _, _, deny, ActReq, EnvReq, ResReq, SubReq).

Rule (21) states that a Policy Set PSN with id PSID that belongs to the Policy Set PPS will have the affect indeterminate on the request with values ActReq, EnvReq, ResReq, SubReq under certain conditions.

The first condition is that there is a Policy Set PSN with id PSID, that belongs to Policy Set PPS and uses the combining algorithm ALG, indicated by the atom [policy(PSN, PSID, PPS, ALG)]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that there is no Policy Set PSN with id PSID that belongs to the Policy Set PPS that has the effect permit on the request with values ActReq, EnvReq, ResReq, SubReq, indicated by the atom [not match_policyset(PSN, PSID, PPS, _, _, permit, ActReq, EnvReq, ResReq, SubReq)]. The fourth condition is that there is no Policy Set PSN with id PSID that belongs to the Policy Set PPS that has the effect deny on the request with values ActReq, EnvReq, ResReq, SubReq, indicated by the atom [not match_policyset(PSN, PSID, PPS, _, _, deny, ActReq, EnvReq, ResReq, SubReq)].

(22)

```
match_program(PSRoot, CN, RN, E, ActReq, EnvReq, ResReq, SubReq) :-
                                    PSRoot = ps0,
                                    request(ActReq, EnvReq, ResReq, SubReq),
                                    match_policyset(PSRoot, _, _, CN, RN, E, ActReq,
                                    EnvReq, ResReq, SubReq).
```

Rule (22) states that the whole program, which is identified as the root Policy Set PSRoot, will have the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN that belongs to child policy CN under certain conditions.

The first condition is that the Policy Set PSRoot is actually the root Policy Set ps0, indicated by the atom [PSRoot = ps0]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that there is a Policy Set PSRoot that has the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN that belongs to child policy CN, indicated by the atom [match_policyset(PSRoot, _, _, CN, RN, E, ActReq, EnvReq, ResReq, SubReq)].

(23)

```
match_policyset_child(CN, CID, CPN, RN, E, ActReq, EnvReq, ResReq, SubReq) :-
```

policy_set(CPN, _, _, _),
policy(CN, CID, CPN, _),
request(ActReq, EnvReq, ResReq, SubReq),
match_policy(CN, CID, CPN, RN, E, ActReq, EnvReq, ResReq, SubReq).

Rule (23) states that a child policy CN of the Policy Set CPN with id CID will have the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN under certain conditions.

The first condition is that there is a Policy Set CPN, indicated by the atom [policy_set(CPN, _, _, _)]. The second condition is that there is a policy CN with id CID which belongs to the Policy Set CPN, indicated by the atom [policy(CN, CID, CPN, _)]. The third condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The fourth condition is that there is a policy CN with id CID that belongs to the Policy Set CPN that has the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN, indicated by the atom [match_policy(CN, CID, CPN, RN, E, ActReq, EnvReq, ResReq, SubReq)].

(24)

match_policyset_child(CN, CID, CPN, RN, E, ActReq, EnvReq, ResReq, SubReq) :-
policy_set(CPN, _, _, _),
policy_set(CN, CID, CPN, _),
CN != CPN,
request(ActReq, EnvReq, ResReq, SubReq),
match_policyset(CN, CID, CPN, _, RN, E, ActReq, EnvReq, ResReq, SubReq).

Rule (24) states that a child policy CN of the Policy Set CPN with id CID will have the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN under certain conditions.

The first condition is that there is a Policy Set CPN, indicated by the atom [policy_set(CPN, _, _, _)]. The second condition is that there is a policy CN with id CID which belongs to the Policy Set CPN, indicated by the atom [policy(CN, CID, CPN, _)]. The third condition is that the child policy CN is not its own parent CPN, indicated by the atom [CN != CPN]. The fourth condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom

[request(ActReq, EnvReq, ResReq, SubReq)]. The fifth condition is that there is a Policy Set CN with id CID that belongs to the Policy Set CPN that has the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN, indicated by the atom [match_policy(CN, CID, CPN, _, RN, E, ActReq, EnvReq, ResReq, SubReq)].

2.3.3 Mapping combining algorithms

In the XACML 3.0 abstract syntax there are four common combining algorithms defined, First-applicable, Permit-overrides, Deny-overrides, and Only-one-applicable.

2.3.3.1 Permit overrides

In the permit-overrides combining algorithm, if there is a permit rule/policy, which matches the request, then the result is Permit; otherwise, the result is determined by deny rules/policies.

(25)

```
match_policy_alg(PN, RN, permit_overrides, permit, ActReq, EnvReq, ResReq, SubReq) :-
                                    policy(PN, _, _, permit_overrides),
                                    request(ActReq, EnvReq, ResReq, SubReq),
                                    match_rule(RN, _, PN, permit, ActReq, EnvReq,
                                    ResReq, SubReq).
```

Rule (25) states that the policy PN using the combining algorithm permit overrides will permit the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN under certain conditions.
The first condition is that policy PN actually uses the combining algorithm permit overrides indicated by the atom [policy(PN, _, _, permit_overrides)]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that there is a matched rule RN that belongs to the policy PN and its effect under the request with values ActReq, EnvReq, ResReq, SubReq is permit, indicated by the atom [match_rule(RN, _, PN, permit, ActReq, EnvReq, ResReq, SubReq)].

(26)

```
match_policy_alg(PN, RN, permit_overrides, deny, ActReq, EnvReq, ResReq, SubReq) :-
```

policy(PN, _, _, permit_overrides),
request(ActReq, EnvReq, ResReq, SubReq),
not match_rule(_, _, PN, permit, ActReq, EnvReq, ResReq, SubReq),
match_rule(RN, _, PN, deny, ActReq, EnvReq, ResReq, SubReq).

Rule (26) states that the policy PN using the combining algorithm permit overrides will deny the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN under certain conditions.

The first condition is that policy PN actually uses the combining algorithm permit overrides indicated by the atom [policy(PN, _, _, permit_overrides)]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that there is no matched rule RN that belongs to the policy PN and its effect under the request with values ActReq, EnvReq, ResReq, SubReq is permit, indicated by the atom [not match_rule(RN, _, PN, permit, ActReq, EnvReq, ResReq, SubReq)]. The fourth condition is that is that there is a matched rule RN that belongs to the policy PN and its effect under the request with values ActReq, EnvReq, ResReq, SubReq is deny, indicated by the atom [match_rule(RN, _, PN, deny, ActReq, EnvReq, ResReq, SubReq)].

(27)

match_policyset_alg(PS, CN, RN, permit_overrides, permit, ActReq, EnvReq, ResReq, SubReq) :-

policy_set(PS, _, _, permit_overrides),
request(ActReq, EnvReq, ResReq, SubReq),
match_policyset_child(CN, _, PS, RN, permit, ActReq, EnvReq, ResReq, SubReq).

(28)

match_policyset_alg(PS, CN, RN, permit_overrides, deny, ActReq, EnvReq, ResReq, SubReq) :-

policy_set(PS, _, _, permit_overrides),
request(ActReq, EnvReq, ResReq, SubReq),

not match_policyset_child(_, _, PS, _, permit, ActReq, EnvReq, ResReq, SubReq),
match_policyset_child(CN, _, PS, RN, deny, ActReq, EnvReq, ResReq, SubReq).

Rules (27) and (28) follow similar logic to rules (25) and (26) respectively. Here we are referring to the effect of Policy Sets based on the decisions of their child policies or child Policy Sets. Whereas in rules (25) and (26) we were referring to the effect of policies based on the decisions of their rules.

2.3.3.2 Deny overrides

In the deny-overrides combining algorithm, if there is a deny rule/policy, which matches the request, then the result is deny; otherwise, the result is obtained by permit rules/policies.

(29)

```
match_policy_alg(PN, RN, deny_overrides, deny, ActReq, EnvReq, ResReq, SubReq) :-
        policy(PN, _, _, deny_overrides),
        request(ActReq, EnvReq, ResReq, SubReq),
        match_rule(RN, _, PN, deny, ActReq, EnvReq, ResReq, SubReq).
```

Rule (29) states that the policy PN using the combining algorithm deny overrides will deny the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN under certain conditions. The first condition is that policy PN actually uses the combining algorithm deny overrides indicated by the atom [policy(PN, _, _, deny_overrides)]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that there is a matched rule RN that belongs to the policy PN and its effect under the request with values ActReq, EnvReq, ResReq, SubReq is deny, indicated by the atom [match_rule(RN, _, PN, deny, ActReq, EnvReq, ResReq, SubReq)].

(30)

```
match_policy_alg(PN, RN, deny_overrides, permit, ActReq, EnvReq, ResReq, SubReq) :-
        policy(PN, _, _, deny_overrides),
```

request(ActReq, EnvReq, ResReq, SubReq),
not match_rule(_, _, PN, deny, ActReq, EnvReq, ResReq, SubReq),
match_rule(RN, _, PN, permit, ActReq, EnvReq, ResReq, SubReq).

Rule (30) states that the policy PN using the combining algorithm deny overrides will permit the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN under certain conditions.

The first condition is that policy PN actually uses the combining algorithm deny overrides indicated by the atom [policy(PN, _, _, deny_overrides)]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that there is no matched rule RN that belongs to the policy PN and its effect under the request with values ActReq, EnvReq, ResReq, SubReq is deny, indicated by the atom [not match_rule(RN, _, PN, deny, ActReq, EnvReq, ResReq, SubReq)]. The fourth condition is that is that there is a matched rule RN that belongs to the policy PN and its effect under the request with values ActReq, EnvReq, ResReq, SubReq is permit, indicated by the atom [match_rule(RN, _, PN, permit, ActReq, EnvReq, ResReq, SubReq)].

(31)

match_policyset_alg(PS, CN, RN, deny_overrides, deny, ActReq, EnvReq, ResReq, SubReq) :-

policy_set(PS, _, _, deny_overrides),
request(ActReq, EnvReq, ResReq, SubReq),
match_policyset_child(CN, _, PS, RN, deny, ActReq, EnvReq, ResReq, SubReq).

(32)

match_policyset_alg(PS, CN, RN, deny_overrides, permit, ActReq, EnvReq, ResReq, SubReq) :-

policy_set(PS, _, _, deny_overrides),
request(ActReq, EnvReq, ResReq, SubReq),
not match_policyset_child(_, _, PS, _, deny, ActReq, EnvReq, ResReq, SubReq),

match_policyset_child(CN, _, PS, RN, permit, ActReq, EnvReq, ResReq, SubReq).

Rules (31) and (32) follow similar logic to rules (29) and (30) respectively. Here we are referring to the effect of Policy Sets based on the decisions of their child policies or child Policy Sets. Whereas in rules (29) and (30) we were referring to the effect of policies based on the decisions of their rules.

2.3.3.3 First applicable

In the first-applicable combining algorithm, the result is determined by the matching result of the first rule/policy whose target and condition are matched to the decision request.

(33)

```
dom_match_rule(RN, RID1, PN, E, ActReq, EnvReq, ResReq, SubReq):-
                request(ActReq, EnvReq, ResReq, SubReq),
                match_rule(RN, RID1, PN, E, ActReq, EnvReq, ResReq, SubReq),
                match_rule(_, RID2, PN, _, ActReq, EnvReq, ResReq, SubReq),
                RID2<RID1.
```

Rule (33) states that the dominated rule RN with id RID1 that belongs to policy PN will have the effect E on the request with values ActReq, EnvReq, ResReq, SubReq under certain conditions.

The first condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The second condition is that there is a matched rule RN with id RID1 that belongs to the policy PN and its effect under the request with values ActReq, EnvReq, ResReq, SubReq is E, indicated by the atom [match_rule(RN, RID1, PN, E, ActReq, EnvReq, ResReq, SubReq)]. The third condition is that there is a matched rule with id RID2 that also belongs to policy PN indicated by the atom [match_rule(_, RID2, PN, _, ActReq, EnvReq, ResReq, SubReq)]. The fourth condition is that id RID1 of rule RN is larger than the id of the other matched rule with id RID2, meaning that rule with id RID1 is dominated by some higher priority rule with id RID2 within the same policy.

(34)

```
match_policy_alg(PN, RN, first_applicable, E, ActReq, EnvReq, ResReq, SubReq) :-
                                policy(PN, _, _, first_applicable),
                                request(ActReq, EnvReq, ResReq, SubReq),
                                match_rule(RN, RID, PN, E, ActReq, EnvReq, ResReq,
                                SubReq),
                                not dom_match_rule(_, RID, PN, E, ActReq, EnvReq,
                                ResReq, SubReq).
```

Rule (34) states that the policy PN using the combining algorithm first applicable will have the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN under certain conditions.

The first condition is that policy PN actually uses the combining algorithm first applicable indicated by the atom [policy(PN, _, _, first_applicable)]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that there is a matched rule RN that belongs to the policy PN and its effect under the request with values ActReq, EnvReq, ResReq, SubReq is E, indicated by the atom [match_rule(RN, RID, PN, E, ActReq, EnvReq, ResReq, SubReq)]. The fourth condition is that the same rule with id RID must not be dominated by some higher priority rule indicated by the atom [not dom_match_rule(_, RID, PN, E, ActReq, EnvReq, ResReq, SubReq)].

(35)

```
dom_match_child(CN, CID1, PS, E, ActReq, EnvReq, ResReq, SubReq):-
                                request(ActReq, EnvReq, ResReq, SubReq),
                                match_policyset_child(CN, CID1, PS, _, E, ActReq,
                                EnvReq, ResReq, SubReq),
                                match_policyset_child(_, CID2, PS, _, E2, ActReq,
                                EnvReq, ResReq, SubReq),
                                E != indeterminate,
                                E2 != indeterminate,
                                CID2<CID1.
```

(36)

```
match_policyset_alg(PS, CN, RN, first_applicable, E, ActReq, EnvReq, ResReq, SubReq) :-
                                        policy_set(PS, _, _, first_applicable),
                                        request(ActReq, EnvReq, ResReq, SubReq),
                                        match_policyset_child(CN, CID, PS, RN, E, ActReq,
                                        EnvReq, ResReq, SubReq),
                                        E != indeterminate,
                                        not dom_match_child(CN, CID, PS, E, ActReq, EnvReq,
                                        ResReq, SubReq).
```

Rules (35) and (36) follow similar logic to rules (33) and (34) respectively. Here we are referring to the effect of Policy Sets based on the decisions of their child policies or child Policy Sets where we also add a condition which states that the effect must not be indeterminate, indicated by the atoms [E != indeterminate, E2 != indeterminate]. Whereas in rules (34) and (35) we were referring to the effect of policies based on the decisions of their rules.

2.3.3.4 Only one applicable

In the only-one-applicable combining algorithm, if exactly one policy is matched, the result of the combining algorithm is identified by such a policy.

(37)

```
match_policy_alg(PN, RN, only_one_applicable, E, ActReq, EnvReq, ResReq, SubReq) :-
                                        policy(PN, _, _, only_one_applicable),
                                        request(ActReq, EnvReq, ResReq, SubReq),
                                        match_rule(RN, _, PN, E, ActReq, EnvReq, ResReq,
                                        SubReq),
                                        1{match_rule(_, _, PN, _, ActReq, EnvReq, ResReq,
                                        SubReq)}1.
```

Rule (37) states that the policy PN using the combining algorithm only one applicable will have the affect E on the request with values ActReq, EnvReq, ResReq, SubReq due to rule RN under certain conditions.

The first condition is that policy PN actually uses the combining algorithm only one applicable indicated by the atom [policy(PN, _, _, only_one_applicable)]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom

[request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that there is a matched rule RN that belongs to the policy PN and its effect under the request with values ActReq, EnvReq, ResReq, SubReq is E, indicated by the atom [match_rule(RN, RID, PN, E, ActReq, EnvReq, ResReq, SubReq)]. The fourth condition is that there is only a single rule that belongs to policy PN that is matched under the request with values ActReq, EnvReq, ResReq, SubReq, indicated by the atom [1{match_rule(_, _, PN, _, ActReq, EnvReq, ResReq, SubReq)}1].

(38)

```
match_policyset_alg(PS, CN, RN, only_one_applicable, E, ActReq, EnvReq, ResReq, SubReq) :-
                                    policy_set(PS, _, _, only_one_applicable),
                                    request(ActReq, EnvReq, ResReq, SubReq),
                                    match_policyset_child(CN, _, PS, RN, E, ActReq, EnvReq, ResReq, SubReq),
                                    E != indeterminate,
                                    1{match_policyset_child(_, _, PS, _, E2, ActReq, EnvReq, ResReq, SubReq) : E2 != indeterminate}1.
```

Rule (38) follows similar logic to rule (37). Here we are referring to the effect of Policy Sets based on the decisions of their child policies or child Policy Sets where we also add a condition which states that the effect must not be indeterminate, indicated by the atom [E != indeterminate]. Whereas rule (37) refers to the effect of policies based on the decisions of their rules.

2.3.4 Mapping requests

In XACML the request component is a set of attribute types that are assigned values relative to their domains, which are defined by the input XACML policy. The XACBench prototype tool defines four attribute types and a domain for each one. This representation of attributes is more in line with XACML 2.0 since it involves the attribute types of action, environment, resource, and subject.

(39)

```
request(ActReq, EnvReq, ResReq, SubReq) :- action_domain(ActReq), environment_domain(EnvReq),
```

resource_domain(ResReq),

subject_domain(SubReq).

Rule (39) states that a request with attribute values ActReq, EnvReq, ResReq, SubReq is created under certain conditions. The first condition is that ActReq belongs to the action domain, indicated by the atom [action_domain(ActReq)]. The second condition is that EnvReq belongs to the environment domain, indicated by the atom [environment_domain(EnvReq)]. The third condition is that ResReq belongs to the resource domain, indicated by the atom [resource_domain(ResReq)]. The fourth condition is that SubReq belongs to the subject domain, indicated by the atom [subject_domain(SubReq)].

In our running example with the real-world XACML 3.0 policy called "kmarket-gold-policy.xml" (Appendix A), we consider a possible request for the policy, shown in Figure 2.3 (Appendix B).

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    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"
    ReturnPolicyIdList="false" CombinedDecision="false">
    <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
        <Attribute IncludeInResult="false"
            AttributeId="http://kmarket.com/id/role">
            <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">gold</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="http://kmarket.com/category">
        <Attribute IncludeInResult="false"
            AttributeId="http://kmarket.com/id/totalAmount">
            <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">550</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
        <Attribute IncludeInResult="false"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
            <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">Liquor</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="http://kmarket.com/category">
        <Attribute IncludeInResult="false"
            AttributeId="http://kmarket.com/id/amount">
            <AttributeValue
            DataType="http://kmarket.com/id/amount">20</AttributeValue>
        </Attribute>
    </Attributes>
</Request>
```

Figure 2.3: XACML 3.0 request for real-world XACML 3.0 policy called "kmarket-gold-policy.xml".

The ASP translation of this request would result in an ASP fact such as:

request(gold, 550, liquor, 20).

2.4 ASP programs of the middle framework layer

The XACBench prototype XACML analysis: total redundancy, simple redundancy, shadow anomaly, correlation anomaly, generalization anomaly, reachability, usefulness, isomorphism, and completeness.

To analyse these XACML policy properties we need ASP programs that describe them using ASP rules. The XACBench prototype tool contains such ASP programs: “all_rules_total_redundancy.asp”, “completeness.asp”, “effectiveness_isomorphism.asp”, “effectiveness_policy.asp”, “effectiveness_policyset.asp”, “effectiveness_rule.asp”, “inter_policy_anomalies.asp”, and “intra_policy_anomalies.asp” , their rules can be seen in Sections 2.4.1-2.4.7.

2.4.1 Reachability

A rule is reachable if there is a request matched by this rule. Removing an unreachable rule (policy and policy set) has no effect on the semantics of a policy. Thus, the discovery of such rules (policies and policy sets) helps the administrator to remove them and improve the efficiency of the policy analysis.

```
reachable_rule(RN) :-
                        rule(RN, _, _, _),
                        request(ActReq, EnvReq, ResReq, SubReq),
                        match_program(_, _, RN, _, ActReq, EnvReq, ResReq,
                        SubReq).

unreachable_rule(RN) :-
                        rule(RN, _, _, _),
                        not reachable_rule(RN).
```

The XACBench tool contains similar predefined ASP programs and ASP rules for reachability and unreachability of policies and policy sets.

2.4.2 Usefulness

A rule (policy, policy set) is useful if it is matched by some request. Removing a useless rule (policy and policy set) has no effect on the semantics of a policy but can improve the efficiency of policy analysis.

A rule is useless if there is no request matched by the rule.

```
useful_rule(RN) :-
                    rule(RN, RID, P, E),
                    request(ActReq, EnvReq, ResReq, SubReq),
                    match_rule(RN, RID, P, E, ActReq, EnvReq, ResReq,
                    SubReq).

useless_rule(RN) :-

                    rule(RN, _, _, _),
                    not useful_rule(RN).
```

The XACBench tool contains similar predefined ASP programs and ASP rules for usefulness of policies and policy sets.

2.4.3 Total redundancy

A rule(policy) is totally redundant if every request matched by this rule(policy) is also matched by other rules(policies) in the policy (Policy Set).

A rule is matched by another rule of the same policy if there is a request matched by both rules.

```
match_by_others(RN, ActReq, EnvReq, ResReq, SubReq) :-
                    rule(RN, RID, PN, _),
                    rule(RN2, RID2, PN, _),
```

```
                    RID > RID2,
                    request(ActReq, EnvReq, ResReq, SubReq),
                    match_rule(RN, RID, PN, _, ActReq, EnvReq, ResReq,
                    SubReq),
                    match_rule(RN2, RID2, PN, _, ActReq, EnvReq,
                    ResReq, SubReq).
```

A rule is not totally redundant if there is a request matched by that rule and no other rule of the same policy is matched by that same request.

```
no_total_redundancy(RN) :-
                    rule(RN, _, _, _),
                    request(ActReq, EnvReq, ResReq, SubReq),
                    match_rule(RN, _, _, _, ActReq, EnvReq, ResReq,
                    SubReq),
                    not match_by_others(RN, ActReq, EnvReq, ResReq,
                    SubReq).

total_redundancy(RN) :-
                    rule(RN, _, _, _),
                    not no_total_redundancy(RN).
```

A policy is matched by another policy of the same Policy Set if there is a request matched by both policies.

```
match_by_others(PN, ActReq, EnvReq, ResReq, SubReq) :-
                    policy(PN, PID, PSN, _),
                    policy(PN2, PID2, PSN, _),
                    PID > PID2,
                    request(ActReq, EnvReq, ResReq, SubReq),
                    match_policy(PN, PID, PSN, _, _, ActReq, EnvReq,
                    ResReq, SubReq),
                    match_policy(PN2, PID2, PSN, _, _, ActReq, EnvReq,
                    ResReq, SubReq).
```

A policy is not totally redundant if there is a request matched by that policy and no other policy of the same Policy Set is matched by that same request.

```
no_total_redundancy(PN) :-
                                policy(PN, PID, PSN, _),
                                request(ActReq, EnvReq, ResReq, SubReq),
                                match_policy(PN, PID, PSN, _, _, ActReq, EnvReq,
                                ResReq, SubReq),
                                not match_by_others(PN, ActReq, EnvReq, ResReq,
                                SubReq).

total_redundancy(PN) :-
                                policy(PN, PID, PSN, _),
                                not no_total_redundancy(PN).
```

2.4.4 Completeness

A XACML policy is incomplete if there is a request that is neither matched as permit nor as deny by the program.

A policy is complete if it matches all possible requests.

```
incomplete(ActReq, EnvReq, ResReq, SubReq) :-
                                request(ActReq, EnvReq, ResReq, SubReq),
                                not match_program(_, _, _, permit, ActReq, EnvReq,
                                ResReq, SubReq),
                                not match_program(_, _, _, deny, ActReq, EnvReq,
                                ResReq, SubReq).

complete :- not incomplete(_, _, _, _).
```

2.4.5 Isomorphism

Two XACML policies are isomorphic if and only if they return identical decision for every request.

Two programs (policysets) are not isomorphic if they have different decisions (E1!=E2) for at least one common request.

```
no_isomorphic_program(PSN1, PSN2) :-
                                    policy_set(PSN1, PSID1, _, ALG),
                                    policy_set(PSN2, PSID2, _, ALG),
                                    PSID1 != PSID2,
                                    request(ActReq, EnvReq, ResReq, SubReq),
                                    match_policyset(PSN1, _, _, _, _, E1, ActReq, EnvReq,
                                    ResReq, SubReq),
                                    match_policyset(PSN2, _, _, _, _, E2, ActReq, EnvReq,
                                    ResReq, SubReq),
                                    E1 != E2.
```

Two programs (policysets) are not isomorphic if one is matched by a common request and the other is not.

```
no_isomorphic_program(PSN1, PSN2) :-
                                    policy_set(PSN1, PSID1, _, ALG),
                                    policy_set(PSN2, PSID2, _, ALG),
                                    PSID1 != PSID2,
                                    request(ActReq, EnvReq, ResReq, SubReq),
                                    match_policyset(PSN1, _, _, _, _, _, ActReq, EnvReq,
                                    ResReq, SubReq),
                                    not match_policyset(PSN2, _, _, _, _, _, ActReq,
                                    EnvReq, ResReq, SubReq).
```

The isomorphism property of two policy sets is symmetric. If PSN2 is isomorphic to PSN1 then PSN1 is isomorphic to PSN2. The same can be said for a non-isomorphic pair of policy sets.

```
isomorphic_program(PSN1, PSN2) :-
                                    isomorphic_program(PSN2, PSN1).

no_isomorphic_program(PSN1, PSN2) :-
```

```
no_isomorphic_program(PSN2, PSN1).
```

Two policy sets are isomporhpic if they are not non-isomporhpic.

```
isomorphic_program(PSN1, PSN2) :-
        policy_set(PSN1, PSID1, _, ALG),
        policy_set(PSN2, PSID2, _, ALG),
        PSID1 != PSID2,
        not no_isomorphic_program(PSN1, PSN2).
```

2.4.6 Intra-policy anomalies

Intra-policy anomalies are anomalies which occur between rules of the same policy. The ASP rules described below check for anomalies of a rule with other rules within a policy. These anomalies consist of simple shadow, simple redundancy, correlation, generalization, and some other anomalies that we described before, such as total redundancy, usefulness, and reachability of a rule.

A rule RID1 is not a subset of another rule RID2 of the same policy if the first matches a common request and the second does not.

```
no_subset_rule(RID1, RID2) :-
        rule(RN1, RID1, PN, _),
        rule(RN2, RID2, PN, _),
        RN1 != RN2,
        request(ActReq, EnvReq, ResReq, SubReq),
        match_rule(RN1, RID1, PN, _, ActReq, EnvReq,
        ResReq, SubReq),
        not match_rule(RN2, RID2, PN, _, ActReq, EnvReq,
        ResReq, SubReq).
```

```
subset_rule(RID1, RID2) :-
        rule(RN1, RID1, PN, _),
        rule(RN2, RID2, PN, _),
        RN1 != RN2,
```

```
                    not no_subset_rule(RID1, RID2).
```

A rule RID1 overlaps with another rule RID2 of the same policy if they both match at least one common request. The overlap property is symmetric, and it helps us verify the simple correlation anomaly.

```
overlap_rule(RID1, RID2) :-
                    overlap_rule(RID2, RID1).

overlap_rule(RID1, RID2) :-
                    rule(RN1, RID1, PN, _),
                    rule(RN2, RID2, PN, _),
                    RN1 != RN2,
                    request(ActReq, EnvReq, ResReq, SubReq),
                    match_rule(RN1, RID1, PN, _, ActReq, EnvReq,
                    ResReq, SubReq),
                    match_rule(RN2, RID2, PN, _, ActReq, EnvReq,
                    ResReq, SubReq).
```

A rule RID1 is shadowed by another rule RID2 of the same policy if the first is a subset of the second one, they have different effects, and the first has a lower priority.

```
simple_shadow_anomaly_rule(RID1, RID2) :-
                    rule(RN1, RID1, PN, RE1),
                    rule(RN2, RID2, PN, RE2),
                    RID2 < RID1,
                    RE1 != RE2,
                    subset_rule(RID1, RID2).
```

A rule RID1 is redundant due to another rule RID2 of the same policy if the first is a subset of the second one, they have the same effects, and the first has a lower priority. Removing a redundant or shadowed rule has no effect on the semantics of a policy but can improve the efficiency of policy analysis.

```
simple_redundancy_anomaly_rule(RID1, RID2) :-
```

```
                rule(RN1, RID1, PN, RE1),
                rule(RN2, RID2, PN, RE2),
                RID2 < RID1,
                RE1 == RE2,
                subset_rule(RID1, RID2).
```

A rule RID1 is a correlation of a rule RID2 of the same policy if they have different effects, the first has lower priority and is overlapped by the second one, and they are not subsets of each other. In other words, there are request that are only matched by the first rule, requests that are only matched by the second rule, and requests that are matched by both rules. If the order (the priority) of the two rules was reversed, the decisions (effects) for the requests that are matched by both rules would also be reversed. Correlation is considered an anomaly warning [10].

```
correlation_anomaly_rule(RID1, RID2) :-
                rule(RN1, RID1, PN, RE1),
                rule(RN2, RID2, PN, RE2),
                RID2 < RID1,
                RE1 != RE2,
                overlap_rule(RID1, RID2),
                not subset_rule(RID1, RID2),
                not subset_rule(RID2, RID1).
```

A rule RID2 is a generalization of a rule RID1 of the same policy if they have different effects, the first has lower priority and is a subset of the second. A generalization rule is used to exclude a particular request from the general evaluation of the policy. Suppose we have a policy with rule R1 that is matched and evaluated to permit by all request coming from surgeons and another rule R2 of the same policy, that is matched specifically by a surgeon with the name “John Hopkins” and its effect is deny. If R1 has a higher priority, then it is considered a generalization of R2. Generalization is considered only an anomaly warning because the specific rule, R2 in this case, makes an exception of the general rule. It is important to highlight its effect to the policy developers for confirmation.

```
simple_generalization_anomaly_rule(RID1, RID2) :-
                rule(RN1, RID1, PN, RE1),
```

```
rule(RN2, RID2, PN, RE2),
RID2 > RID1,
RE1 != RE2,
subset_rule(RID1, RID2).
```

2.4.7 Inter-policy anomalies

Inter-policy anomalies are anomalies which occur between policies of the same Policy Set. These anomalies, like intra-policy anomalies, consist of simple shadow, simple redundancy, correlation, generalization, total redundancy, usefulness, and reachability of a policy. The XACBench tool contains ASP programs and ASP rules similar those in the previous Section 2.4.7 but in this case, they are applied to policies of same policy sets. The motivation for the verification of these properties is the improvement of policy analysis through removal of unnecessary policies.

2.5 ASP request generator

To analyse an XACML policy for specific properties (Section 2.4), we need to generate all possible requests for that policy. XACBench contains an ASP program called "generate_all.asp" which represents the request generator. The first four lines represent the domains of the four standard attribute types (Action, Environment, Resource, and Subject). Each domain is a set of values defined by a policy administrator, which in this case are the developers of XACBench. Rule (39) from Section 2.3.1 creates a request for each combination of the values of the attribute type domains.

```
action_domain( act_value1; act_value2; act_value3; …).
environment_domain( env_value1; env_value2; env_value3; …).
resource_domain( res_value1; res_value2; res_value3; …).
subject_domain(sub_value1; sub_value2; sub_value3; …).

request(ActReq, EnvReq, ResReq, SubReq) :- action_domain(ActReq), environment_domain(EnvReq), resource_domain(ResReq), subject_domain(SubReq).
```

Chapter 3

Modifying XACBench

3.1 Introduction 61

3.2 Arbitrary attribute types of XACML 3.0 62

3.3 Conditions of XACML 3.0 64

3.4 Automatic target matching 69

3.5 Reducing atom arity for query analysis 71

3.6 Authorization hierarchy 74

3.1 Introduction

Although XACBench (Section 2.2.2) is a useful prototype tool that helps us generate synthetic XACML policies, its XACML 3.0 to ASP translation functionality is incomplete. In this section we will present some problems in XACBench that stop us from being able to correctly analyze properties of XACML 3.0 policies. There are limitations imposed by both the predefined ASP programs and by the java code that is responsible for translating the input XACML policies into equivalent ASP programs. To overcome these issues, we propose and implement a variety of adjustments. These adjustments help us generate accurate translations of the input XACML 3.0 policies and in turn let us proceed with XACML 3.0 policy analysis. In this section we also propose a way to improve the query analysis process and a new ASP program which will give us an insight into the authorization hierarchies of attribute type values within XACML 3.0 policies.

3.2 Arbitrary attribute types of XACML 3.0

As we have mentioned in Section 1.2.5, we have chosen to work with a tool that translates XACML 3.0 policies to ASP equivalent programs because XACML 3.0 allows us to define custom arbitrary attribute types which makes it more expressive than its previous version, XACML 2.0. The problem we found in XACBench, is that this expressiveness is neglected because all of its predefined ASP programs ("all_rules_total_redundancy.asp", "completeness.asp", "effectiveness_isomorphism.asp", "effectiveness_policy.asp", "effectiveness_policyset.asp", "effectiveness_rule.asp", "inter_policy_anomalies.asp", and "intra_policy_anomalies.asp", "match_common.asp", "match_target_v3.asp") are policy independent, which means that they do not adapt to the XACML 3.0 policy given as input. The number of arguments(arity) of the atoms in the predefined ASP programs is fixed, specifically there are four dedicated attribute types indicated by the sequence ActReq, EnvReq, ResReq, SubReq. On the other hand, the number of arguments(arity) of the atoms in the policy dependent ASP program generated by the translation java code, fluctuates based on the number of attributes found by the translator's parser when it goes through the input XACML 3.0 policy. This number is arbitrary since XACML 3.0 policies have an arbitrary number of attribute types. The difference in the number of arguments(arity) and adaptability means that the rules within the generated policy dependent ASP translation program, which are facts that describe the input XACML policy, cannot be matched to the atoms within the bodies of the rules in the predefined policy independent ASP programs. Without common atom arities we are incapable of producing new atoms such as program, policy, rule, target decisions and without these atoms we cannot perform policy analysis.

In Figure 3.1 we have the predefined policy independent ASP program "match_target_v3.asp", which contains ASP rules for matching AnyOf, AllOf, and Target components of XACML 3.0. In Figure 3.2 we have a fragment of the policy dependent ASP translation program generated by the XACBench translator, with the input policy being "continue-a.xml". In this fragment we can see that the atom with the anyof predicate has 15 parameters, 13 of which are dedicated to the attribute types found in "continue-a.xml" by the parser. On the other hand, in the predefined programs, the atoms with the predicate anyof, which is needed for the matching of an AnyOf component to a request in "match_target_v3.asp", has 6 parameters, 4 of which are dedicated to attribute types.

```
match_anyof_req(RN, AnyID, ActReq, EnvReq, ResReq, SubReq) :-
                    request(ActReq, EnvReq, ResReq, SubReq),
                    anyof(RN, AnyID, ActAnyof, EnvAnyof, ResAnyof, SubAnyof),
                    @match_str_func(ActReq, ActAnyof, any) == 1,
                    @match_str_func(EnvReq, EnvAnyof, any) == 1,
                    @match_str_func(ResReq, ResAnyof, any) == 1,
                    @match_str_func(SubReq, SubAnyof, any) == 1.

no_match_anyof_req(RN, ActReq, EnvReq, ResReq, SubReq) :-
                    request(ActReq, EnvReq, ResReq, SubReq),
                    anyof(RN, AnyID, _, _, _, _),
                    not match_anyof_req(RN, AnyID, ActReq, EnvReq, ResReq, SubReq)

match_target(TN, ActReq, EnvReq, ResReq, SubReq) :-
                    request(ActReq, EnvReq, ResReq, SubReq),
                    target(TN).

match_target(TN, ActReq, EnvReq, ResReq, SubReq) :-
                    request(ActReq, EnvReq, ResReq, SubReq),
                    anyof(TN, _, _, _, _, _),
                    not no_match_anyof_req(TN, ActReq, EnvReq, ResReq, SubReq).
```

Figure 3.1: XACBench predefined policy independent ASP program “match_target_v3.asp”.

```
anyof(p1, any2, any, any, any, any, any, any, any, any, any, any, admin, any, any).

anyof(p1, any3, any, any, any, any, any, any, any, any, any, any, any, any, read).
anyof(p1, any3, any, any, any, any, any, any, any, any, any, any, any, any, write).
```

Figure 3.2: Fragment of the “continue-a.xml” policy translation ASP program.

In our modified version of XACBench we overcome this problem by turning the predefined policy independent ASP programs into policy dependent programs. When our modified java translator reads the input XACML policy, it does not only generate a translation ASP program of the input XACML policy, but it also generates a new set of ASP programs using the predefined ASP programs and the arguments found by the parser. This new set of programs contains atoms whose arguments are derived from the input XACML policy. With this adjustment the number of arguments of atoms among all ASP programs is the same and we can run the solver on the generated XACML policy translation and the new set of ASP programs derived from the predefined ones.

As an example, we show in Figure 3.3 one of the ASP programs “kmarket-sliver-policy_match_target_v3.asp” of our new set of programs generated from the predefined policy independent “match_target_v3.asp” ASP program of Figure 3.1.

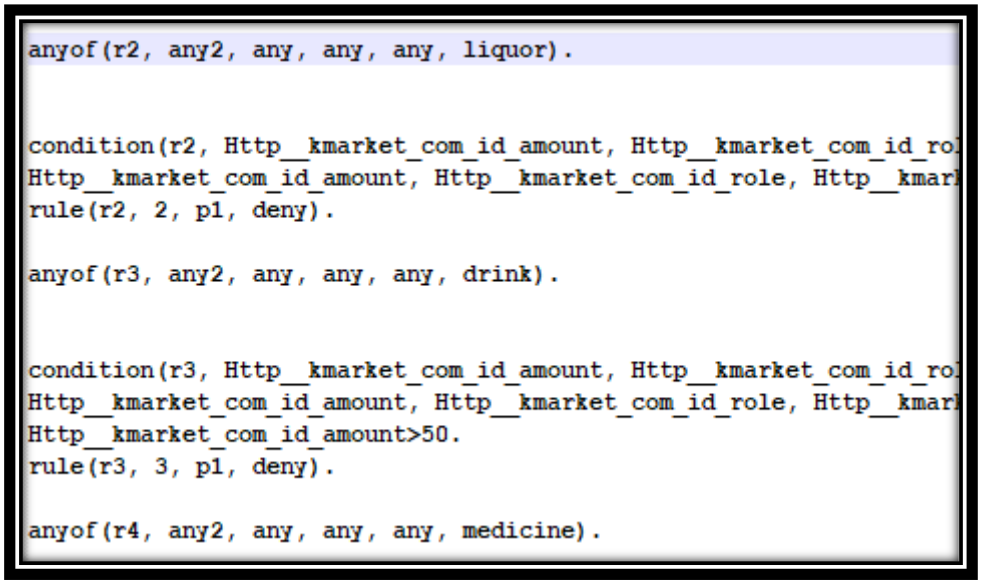
```
match_anyof_req(RN, AnyID, Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id) :-
                request(Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id),
                anyof(RN,
                AnyID,Http__kmarket_com_id_amountAnyof,Http__kmarket_com_id_roleAnyof,Http__kmarket_com_id_totalamountAnyof,Urnoasisnamestcxacml1_0resourceresource_id
                Anyof),
                    @match_str_func(Http__kmarket_com_id_amount,Http__kmarket_com_id_amountAnyof, any) == any,
                    @match_str_func(Http__kmarket_com_id_role,Http__kmarket_com_id_roleAnyof, any) == any,
                    @match_str_func(Http__kmarket_com_id_totalamount,Http__kmarket_com_id_totalamountAnyof, any) == any,
                    @match_str_func(Urnoasisnamestcxacml1_0resourceresource_id,Urnoasisnamestcxacml1_0resourceresource_idAnyof, any) == any.

no_match_anyof_req(RN, Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id) :-
                request(Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id),
                anyof(RN, AnyID, _, _, _, _),
                not match_anyof_req(RN, AnyID, Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmarket_com_id_totalamount,
                Urnoasisnamestcxacml1_0resourceresource_id).

match_target(TN, Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id) :-
                request(Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id),
                target(TN).

match_target(TN, Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id) :-
                request(Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id),
                anyof(TN, _, _, _, _, _),
                not no_match_anyof_req(TN, Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmarket_com_id_totalamount,
                Urnoasisnamestcxacml1_0resourceresource_id).
```

Figure 3.3: Policy dependent ASP program generated from “match_target_v3.asp”.



```
anyof(r2, any2, any, any, any, liquor).


condition(r2, Http__kmarket_com_id_amount, Http__kmarket_com_id_ro
Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmar
rule(r2, 2, p1, deny).

anyof(r3, any2, any, any, any, drink).


condition(r3, Http__kmarket_com_id_amount, Http__kmarket_com_id_ro
Http__kmarket_com_id_amount, Http__kmarket_com_id_role, Http__kmar
Http__kmarket_com_id_amount>50.
rule(r3, 3, p1, deny).

anyof(r4, any2, any, any, any, medicine).
```

Figure 3.4: Fragment of the ASP translation program of “kmarket-sliver-policy.xml”.

The atoms with the AnyOf predicates in both ASP programs in Figures 3.3 and 3.4 have the same number of arguments and can now be matched to create new atoms.

3.3 Conditions of XACML 3.0

As we have mentioned before, a rule’s condition is a Boolean expression that refines the rule’s applicability beyond its target and in an ASP program the condition component can be evaluated merely using facts and constraints. The syntax for condition translation in the predefined ASP programs of XACBench is described by rules (15-19) in Section 2.3.2. Again, we see that these rules neglect XACML 3.0’s expressiveness by bounding the possible Boolean expressions to two simple predicates (is_subject, is_subject_resource) which can only check

for equality of values of the subject and resource attribute types as shown in Figure 3.5.

```
bool_expr(true).
bool_expr(true, ActReq, EnvReq, ResReq, SubReq) :-
                    request(ActReq, EnvReq, ResReq, SubReq).
bool_expr(is_subject(X), ActReq, EnvReq, ResReq, SubReq) :-
                    condition(_, is_subject(X)),
                    request(ActReq, EnvReq, ResReq, SubReq),
                    X == SubReq.
bool_expr(is_subject_resource(X,Y), ActReq, EnvReq, ResReq, SubReq) :-
                    condition(_, is_subject_resource(X, Y)),
                    request(ActReq, EnvReq, ResReq, SubReq),
                    X == SubReq,
                    Y == ResReq.

match_rule(RN, RID, PN, E, ActReq, EnvReq, ResReq, SubReq) :-
                    rule(RN, RID, PN, E),
                    request(ActReq, EnvReq, ResReq, SubReq),
                    match_target(RN, ActReq, EnvReq, ResReq, SubReq),
                    condition(RN, B),
                    bool_expr(B, ActReq, EnvReq, ResReq, SubReq).
```

Figure 3.5: Fragment of the predefined policy independent ASP program "match_common.asp".

With XACML 3.0, attribute types are arbitrary, and the functions used by conditions are much more diverse and offer more than just checking for equality as shown in Figure 3.6 where the attribute type is http://kmarket.com/id/totalAmount and the function is integer-greater-than. The fixed rules in Figure 3.5 are incapable of describing XACML 3.0s expressiveness regarding conditions.

```
<Condition>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
      <AttributeDesignator AttributeId="http://kmarket.com/id/totalAmount" Category="http://kmarket.com/category" DataType=
      "http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true"/>
    </Apply>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">500</AttributeValue>
  </Apply>
</Condition>
```

Figure 3.6: Fragment of the XACML 3.0 policy "kmarket-sliver-policy.xml".

Apart from the limitations of the predefined ASP programs, there are some modifications needed in the java code of XACBench responsible for the translation of XACML 3.0 policies into equivalent ASP programs.

```java
public String translateToASP(String containerId) {
    // TODO Auto-generated method stub
    String aspProgStr = "";
    aspProgStr += "condition(" +  containerId + ", true).";
    return aspProgStr;
}
```

Figure 3.7: Snippet of java code in XACBench responsible for translating XACML 3.0 Conditions.

As shown in Figure 3.7, during parsing XACBench translates each rule's condition using rule (4) from Section 2.3.1. Which is equivalent to saying that all rules within the input policy have an empty condition. In some cases, this may lead to inaccurate policy analysis results.

In our modified version of XACBench we address this issue by defining a new syntax and by changing the parsing process. We introduce a new way to describe the condition component of XACML 3.0 using ASP rules.

First, we adjust the parsing process. Unlike XACBench, where the parser would only look for attribute types used in target components, we parse the XACML input policy and find all attribute types and their values within target and condition xml element tags. This creates an attribute mapper, a HashMap, which maps attribute types of targets and conditions found in the policy to a list with all values associated to that attribute type. The new list of attribute types defines the atom parameters of our set of ASP programs that will be used to analyse the policy. We then parse the input XACML file again and during this parsing we define new policy Dependent rules that accurately describe conditions of XACML 3.0 policies. Figure 3.8 exhibits an example of such a rule, which depicts the condition in Figure 3.6.

```
condition(r1, Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id):-
request( Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id),
Http__kmarket_com_id_totalamount>500.
```

Figure 3.8: Fragment of the "kmarket-sliver-policy.xml" policy translation ASP program with an example of the newly defined condition matching.

The newly defined syntax for translating XACML 3.0 Conditions to ASP rules such as the one in Figure 3.8 is as follows:

condition(RN, Attr1, Attr2, …, AttrN):-
request(Attr1, Attr2, …, AttrN),
expr1, expr2, …, exprK.

The rule above states that the condition element of rule RN is matched by a request with values Attr1, Attr2, …, AttrN for the N attribute types, which are defined by the attribute type list created after the first parsing, under certain conditions.
The first condition is that there is a request with values Attr1, Attr2, …, AttrN, indicated by the atom [request(Attr1, Attr2, …, AttrN)]. The rest of the conditions are defined by the atoms expr1, expr2, …, exprK, which are boolean expression/constraints that depend on the condition of the rule of the input XACML policy. These expressions have the from <attribute type value assigned by request><function type><attribute type value defined by policy>.

When a rule RN has an empty condition element, the rule generated is as follows:

condition(RN, Attr1, Attr2, …, AttrN):-
request(Attr1, Attr2, …, AttrN).

As for the predefined ASP programs, we remove rules (14-17) and then we modify rule (18) from Section 2.3.2, shown in Figure 3.9, by removing the atom with the bool_expr predicate. The result is a new rule shown in Figure 3.10.

```
match_rule(RN, RID, PN, E, ActReq, EnvReq, ResReq, SubReq) :-
                    rule(RN, RID, PN, E),
                    request(ActReq, EnvReq, ResReq, SubReq),
                    match_target(RN, ActReq, EnvReq, ResReq, SubReq),
                    condition(RN, B),
                    bool_expr(B, ActReq, EnvReq, ResReq, SubReq).
```

Figure 3.9: Modified XACBench Rule matching.

```
match_rule(RN, RID, PN, E, ActReq, EnvReq, ResReq, SubReq) :-
                    rule(RN, RID, PN, E),
                    request(ActReq, EnvReq, ResReq, SubReq),
                    match_target(RN, ActReq, EnvReq, ResReq, SubReq),
                    condition(RN, ActReq, EnvReq, ResReq, SubReq).
```

Figure 3.10: Modified Rule matching.

The rule in Figure 3.10 states that a rule RN with id RID that belongs to the policy PN will have the affect E on the request with values ActReq, EnvReq, ResReq, SubReq under certain conditions.

The first condition is that there is a rule RN with id RID, and effect E, that belongs to policy PN, indicated by the atom [rule(RN, RID, PN, E)]. The second condition is that there is a request with values ActReq, EnvReq, ResReq, SubReq indicated by the atom [request(ActReq, EnvReq, ResReq, SubReq)]. The third condition is that the target of rule RN is matched to a request with values ActReq, EnvReq, ResReq, SubReq, indicated by the atom [match_target(RN, ActReq, EnvReq, ResReq, SubReq)]. The fourth condition is that the rule RN's condition element is matched by the request with values ActReq, EnvReq, ResReq, SubReq, indicated by the atom [condition(RN, ActReq, EnvReq, ResReq, SubReq)] which is generated by the newly defined input policy dependent rule:

```
condition(RN, Attr1, Attr2, …, AttrN):-
request(Attr1, Attr2, …, AttrN),
expr1, expr2, …, exprK.
```

Suppose we have a real-world XACML 3.0 policy called "kmarket-gold-policy.xml" in Appendix A. The modified XACBench tool will generate a policy dependent ASP translation program as follows:

```
1   effect(permit;deny;indeterminate).
2   comb_alg(deny_overrides;permit_overrides;first_applicable;only_one_applicable).
3   comb_alg(do;po;fa;ooa).
4   bool_expr(true).
5   target(r1).
6   condition(r1, Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
7   Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id):-
8   request( Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
9   Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id),
10  Http__kmarket_com_id_totalamount>1000.
11  rule(r1, 1, p1, deny).
12  anyof(r2, any2, any, any, any, liquor).
13  condition(r2, Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
14  Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id):-
15  request( Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
16  Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id),
17  Http__kmarket_com_id_amount>10.
18  rule(r2, 2, p1, deny).
19  target(r3).
20  condition(r3, Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
21  Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id):-
22  request( Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
23  Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id).
24  rule(r3, 3, p1, permit).
25  anyof(p1, any2, any, gold, any, any).
26  policy(p1, 1, ps0, deny_overrides).
27  target(ps0).
28  policy_set(ps0, 0, ps0, first_applicable).
```

In lines 6-10, we use our newly defined ASP rule for the Condition of the first Rule of the Policy (lines 21-30 of Appendix A). In lines 13-17, we use our newly defined ASP rule for the Condition of the second Rule of the Policy (lines 53-362 of Appendix A). Finally in lines 20-24, we use our newly defined ASP rule for the empty Condition of the third Rule of the Policy (lines 72 of Appendix A), indicated by the missing <Condition> xml tag.

3.4 Automatic target matching

In this section we go over the rule (3) from Section 2.3.1. This rule belongs to the policy dependent rules which means that it is generated by the XACBench java code during the translation of the input XACML policy. During the parsing phase, XACBench checks for each of the main components of XACML (rule, policy, Policy Set) whether they have and empty target element. If a component does have an empty target, the XACBench translator generates a fact rule such as rule (2) from Section 2.3.1:

target(rname).

Otherwise, if a component does not have an empty target, meaning that it has at least one AnyOf element, the XACBench translator generates a set of fact rules such as rule (1) from Section 2.3.1:

anyof(rname, anyid, action_value, environment_value, resource_value, subject_value).

This set of rules depends on the number of AnyOf elements that the target of the component has, and the number of AllOf elements within each one of them. Notice that if the target is not empty at least one fact rule with predicate AnyOf will be generated.

Together with the fact rules above XACBench also generates a second rule such as rule (3) from Section 2.3.1:

target(rname) :- anyof(rname ,anyid, _, _).

Generating both of these rules for targets which are not empty, creates an automatic matching of the anyof fact to the body of the rule above. This matching in turn generates the atom target(rname), which means that the target of the component rname is empty. This atom together with rule (12) from Section 2.3.2

```
match_target(TN, ActReq, EnvReq, ResReq, SubReq) :-
                                    request(ActReq, EnvReq, ResReq, SubReq),
                                    target(TN).
```

generates the atom match_target. This means that we have matched a target to a request without checking if the request matches the target's requirements. With this logic, any request can bypass rules (10,11,13) from Section 2.3.2, which is equivalent to saying that any request can match any policy's target element.

We overcome this issue by removing rule (3) of Section 2.3.1 from the XACML to ASP translation syntax and by adjusting code within XACBench so that it does not generate such rules when the Targets of elements are not empty. The result is a policy dependent ASP translation program with no rules such as (3). Appendix D is an example of such the translation

of the real-world XACML 3.0 policy called “kmarket-gold-policy.xml”. We can see that this new translation has no rules of type (3) despite having Target elements.

3.5 Reducing atom arity for query analysis

As we have previously stated in Section 2.5, to verify properties of a XACML 3.0 policy, we need to generate all possible requests for that policy in a single answer set. This stands true when the goal is to verify specific policy properties, like the ones in Section 2.4. But when it comes to query analysis (checking what decision the policy comes to when faced with some request), we do not need to generate all requests at once in a single answer set. With this in mind, we decided to create variant of our initial ASP programs generated by the modified XACBench tool. In this variation we will have a separate answer set for each request that can be created through the combination of attribute type values. Each answer set and all of its atoms will refer to the single request which exists within the answer set. This will allow us to reduce the arity of atoms within the ASP programs by removing parameters that indicated values given to attribute types by the request. The reduced arity of atoms should lead to a smaller number of ground rules and atoms, faster grounding and solving, which in turn means more efficient query analysis.

As an example, we demonstrate how a rule defined in our previous ASP programs will look like in the variation set of ASP programs.

Suppose we have rule from Figure 3.9:

```
match_rule(RN, RID, PN, E, ActReq, EnvReq, ResReq, SubReq) :-
                                    rule(RN, RID, PN, E),
                                    request(ActReq, EnvReq, ResReq, SubReq),
                                    match_target(RN, ActReq, EnvReq, ResReq, SubReq),
                                    condition(RN, ActReq, EnvReq, ResReq, SubReq).
```

The variant set of ASP programs will contain an equivalent rule defined as:

```
match_rule(RN, RID, PN, E) :-
                                    rule(RN, RID, PN, E),
                                    match_target(RN),
```

condition(RN).

If our solver generates an answer set that contains the fact match_rule(RN, RID, PN, E), we know to which request the rule RN was matched to since there is only one request within the answer set. On the other hand if the solver generates an answer set that does not contain the fact match_rule(RN, RID, PN, E) we know that the request within that answer set could not be matched to rule RN of the access policy.

Other rules and their original ASP definition follow similar logic to the rule above when it comes to our variation set of ASP programs. The sequence of parameters reserved for attribute types is removed from all atoms. As another example, suppose we have the following rule found in one of the ASP translation programs of our modified tool:

```
match_policyset(PSN, PSID, PPS, CN, RN, E, A, B, C, D, F, G) :-
                policy_set(PSN, PSID, PPS, ALG),
                request(A, B, C, D, F, G),
                match_target(PSN, A, B, C, D, F, G),
                match_policyset_child(CN, _, PSN, RN, E, A, B, C, D, F, G),
                match_policyset_alg(PSN, CN, RN, ALG, E, A, B, C, D, F, G).
```

Where A, B, C, D, F, G are a sequence of parameters reserved for attribute types of the input policy. The variation of this rule is as follows:

```
match_policyset(PSN, PSID, PPS, CN, RN, E) :-
                policy_set(PSN, PSID, PPS, ALG),
                match_target(PSN),
                match_policyset_child(CN, _, PSN, RN, E),
                match_policyset_alg(PS, CN, RN, ALG, E).
```

Notice we have removed the sequence of A, B, C, D, F, G from all atoms, reduction in arity. For us to have an answer set for each possible request we will create a separate request generator called “generate_one.asp”. Both request generators, “generate_all.asp” and “generate_one.asp” will have the same attribute type domains with the same values within those domains, which are defined by the HashMap created during parsing of the input XACML policy.

The general form of the ASP program “generate_all.asp” for the request generator is as follows:

```
attr1_domain( Attr1_value1; Attr1_value2; Attr1_value3; …).
attr2_domain( Attr2_value1; Attr2_value2; Attr2_value3; …).
…
attrk_domain(Attrk_value1; Attrk_value2; Attrk_value3; …).
request(Attr1Req,Attr2Req,…,AttrkReq) :-

                                        attr1_domain(Attr1Req),
                                        attr2_domain(Attr2Req),
                                        …,
                                        attrk_domain(AttrkReq).
```

The general form of the ASP program "generate_one.asp" for the request generator is as follows:

```
attr1_domain( Attr1_value1; Attr1_value2; Attr1_value3; …).
attr2_domain( Attr2_value1; Attr2_value2; Attr2_value3; …).
…
attrk_domain(Attrk_value1; Attrk_value2; Attrk_value3; …).
1{pick_attr1_domain(X): attr1_domain(X)}1.
1{pick_attr2_domain(X): attr2_domain(X)}1.
…
1{pick_attrk_domain(X): attrk_domain(X)}1.

request(Attr1Req,Attr2Req,…,AttrkReq) :-
                                        pick_attr1_domain(Attr1Req),
                                        pick_attr2_domain(Attr2Req),
                                        …,
                                        pick_attrk_domain(AttrkReq).
```

3.6 Authorization hierarchy

A hierarchy is commonly known as a system in which members of a group are ranked according to relative status. In this thesis we specify our own definition of a hierarchy for XACML, where the system is a XACML policy, the members are values associated with attribute type, the

groups are the attribute types, and the values are ranked based on the range of access and authorization they have on different types of requests.

In our extended version of the XACBench tool, we have added a new functionality which generates an ASP program that helps with the observation of authorization hierarchies for each attribute type of a given input XACML 3.0 policy.

For better understanding, we give a high-level description of a simple made-up XACML policy that specifies which employees of a hospital have access to which recourses of the hospital. Suppose the employees, are either doctors, nurses, or ITs, and the resources are medical files, database, and patient records. The doctors are permitted to access both medical files and patient records but are denied access to the database. The nurses are permitted to access medical files but are denied access to both the database and the patient records. Finally, the IT is permitted to access the database, but there is no clarification to whether he can access patient records or medical files.

The adjusted tool we have described so far will parse the XACML policy and create a list of attribute types, which in this case are SubReq and ResReq. The domain of the attribute type SubReq, which represents the employees of the hospital, will have the values nurse, doctor, and IT, while the domain of the attribute type ResReq, which represent the resources of the hospital, will have the values medical_file, patient_record, and database.

Without the added functionality, the tool would generate all the standard ASP programs that are needed for policy analysis. Then the Clingo solver given the ASP programs as input, would produce an answer set which would contain the following atoms:

match_program(psroot, cn, rn, permit, doctor, patient_record).
match_program(psroot, cn, rn, permit, doctor, medical_file).
match_program(psroot, cn, rn, deny, doctor, database).
match_program(psroot, cn, rn, deny, nurse, patient_record).
match_program(psroot, cn, rn, permit, nurse, medical_file).
match_program(psroot, cn, rn, deny, nurse, database).
match_program(psroot, cn, rn, indeterminate, it, patient_record).
match_program(psroot, cn, rn, indeterminate, it, medical_file).
match_program(psroot, cn, rn, permit, it, database).

With the added functionality, the tool will now generate an additional ASP program with the following rules:

```
no_hier(SubReq2 , SubReq1, subreq) :-
                    match_program(PSRoot, CN, RN, permit, ResReq, SubReq1),
                    match_program(PSRoot, CN, RN, deny, ResReq, SubReq2),
                    request(ResReq, _).

no_hier(SubReq2 , SubReq1, subreq) :-
                    match_program(PSRoot, CN, RN, permit,ResReq, SubReq1),
                    match_program(PSRoot,CN,RN,indeterminate,ResReq,
                    SubReq2),
                    request(ResReq, _).
```

The two rules above state that the subject SubReq2 is not hierarchically above subject SubReq1 if subject SubReq1 is permited access by the policy under some specific request (access to resource ResReq) and subject SubReq2 is either denied access or the access cannot be determined by the policy under that same request.

```
hier(SubReq1, SubReq2, subreq) :-
                    match_program(PSRoot, CN, RN, permit, ResReq, SubReq1),
                    match_program(PSRoot, CN, RN, indeterminate, ResReq,
                    SubReq2),
                    not no_hier(SubReq1, SubReq2, subreq).

hier(SubReq1, SubReq2, subreq) :-
                    match_program(PSRoot, CN, RN, permit, ResReq, SubReq1),
                    match_program(PSRoot, CN, RN, deny, ResReq, SubReq2),
                    not no_hier(SubReq1, SubReq2, subreq).
```

The two rules above state that the subject SubReq1 is hierarchically above subject SubReq2 under certain conditions. The first condition is that subject SubReq1 is permited access by the policy to at least one request (access to resource ResReq) and subject SubReq2 is either denied

access or the access cannot be determined by the policy under that same request. The second condition is that it is not the case that SubReq1 is not hierarchically above subject SubReq2.

```
top_hier(SubReq, subreq):-    match_program(PSRoot, CN, RN, permit, ResReq, SubReq),

                              not hier(_,SubReq, subreq).
```

The rule above states that subject SubReq is on top of a hierarchy under certain conditions. The first condition is that the subject SubReq must be permitted access by the policy to at least one request, because we consider that a value that has no permissions cannot be on top of a hierarchy even none of the values have permissions either. The second condition is that subject SubReq is not hierarchically below any other subject.

The five rules described above are used to determine hierarchy among the values of the attribute type SubReq and the five rules below which follow similar logic to the ones above are instead used to determine hierarchy among the values of the attribute type ResReq.

```
no_hier(ResReq2 , ResReq1, resreq) :-
                              match_program(PSRoot, CN, RN, permit, ResReq1, SubReq),
                              match_program(PSRoot, CN, RN, deny, ResReq2, SubReq),
                              request( _, SubReq).

no_hier(ResReq2 , ResReq1, resreq) :-
                              match_program(PSRoot, CN, RN, permit,ResReq1, SubReq),
                              match_program(PSRoot, CN, RN, indeterminate,ResReq2,
                              SubReq),
                              request(_, SubReq).

hier(ResReq1, ResReq2, resreq) :-
                              match_program(PSRoot, CN, RN, permit, ResReq1, SubReq),
                              match_program(PSRoot, CN, RN, indeterminate, ResReq2,
                              SubReq),
                              not no_hier(ResReq1, ResReq2, resreq).
```

```
hier(ResReq1, ResReq2, resreq) :-
                    match_program(PSRoot, CN, RN, permit, ResReq1, SubReq),
                    match_program(PSRoot, CN, RN, deny, ResReq2, SubReq),
                    not no_hier(ResReq1, ResReq2, resreq).

top_hier(ResReq, resreq):-  match_program(PSRoot, CN, RN, permit, ResReq, SubReq),
                    not hier(_,ResReq, resreq).
```

Now the Clingo solver given the set of ASP programs which contain the newly generated ASP program with the hierarchy rules will produce an answer set with the following atoms:

```
top_hier(medical_file,subreq)
top_hier(database,subreq)
hier(medical_file,patient_record,subreq)
top_hier(doctor,resreq)
top_hier(it,resreq)
hier(doctor,nurse,resreq)
```

The first three atoms describe the hierarchy of the values of the attribute type SubReq and the next three atoms describe the hierarchy of the values of the attribute type ResReq. We can use these atoms to build two graphs, shown in Figure 3.11 and Figure 3.12, that represent the hierarchies of the input XACML policy attribute types SubReq and ResReq respectively. The arrows point towards values of attributes whose permissions are a subset of the values of attributes from which the arrow starts. Nodes with no arrows are on top of the hierarchy.

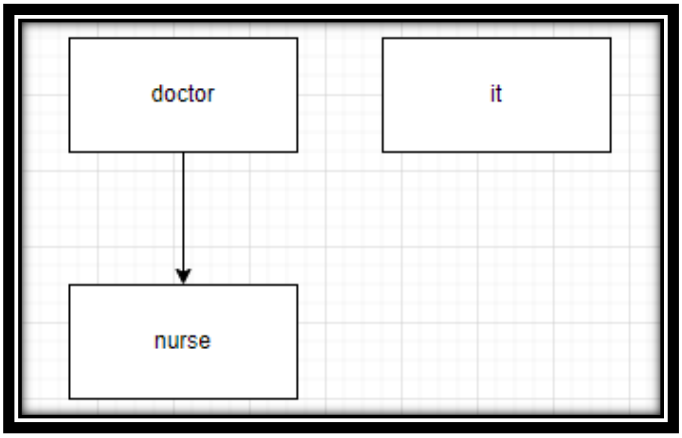


Figure 3.11: Hospital employee hierarchy graph.

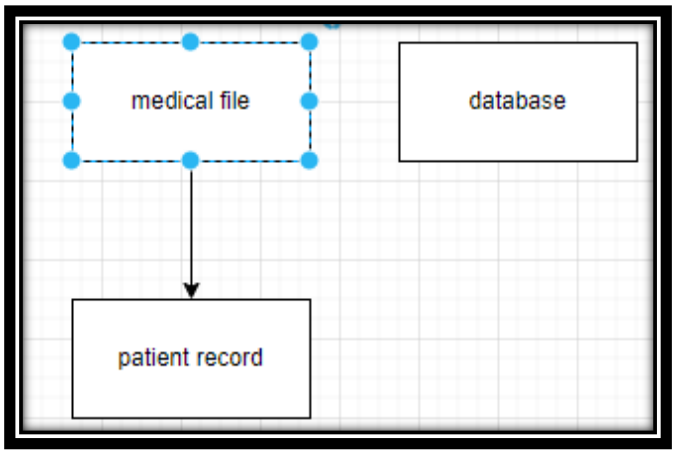


Figure 3.12: Hospital resource hierarchy graph.

This example was very simple, the policy contained only two attribute types with three values in each type. Real life XACML 3.0 policies are made up of much more complex authorization schemes, with many more attribute types and values making it impossible for the human eye to see such hierarchies manually. Our added functionality allows policy developers and analysers to get insight into the hierarchical structure of complex XACML 3.0 policies in a more visually comprehensive way.

Chapter 4

Experiments and Results

4.1 Introduction 80
4.1.1 Experimental environment 81
4.2 Policy property analysis 81
4.2.1 Effectiveness 81
4.2.1.1 Results 82
4.2.2 Efficiency 83
4.2.2.1 Results 85
4.3 Efficiency of variation ASP program for query analysis 87
4.3.1 Results 88
4.4 Hierarchy 89

4.1 Introduction

In this chapter we present how we conducted experiments on our extended version of the XACBench prototype tool and the results we got regarding both the adjustments to the XACBench java code, predefined ASP programs and the additional functionalities that we have implemented.

At first, we go over experiments and results regarding policy property verification, where we examine the tool's effectiveness and efficiency for property analysis. Then we present experiments and results concerning our proposed variation ASP programs which we suggested

as a way to speed up query analysis. And our final experiment demonstrates how our added ASP hierarchy program gives insight into the hierarchy of a real-world policy.

4.1.1 Experimental environment

All experiments were conducted on a AMD Ryzen 7 3700X 8-Core Processor3.59 GHz PC with 8 GB of RAM running on Windows 10. Clingo 5.5.0 was used for both the grounding and solving of the ASP programs.

4.2 Policy property analysis

4.2.1 Effectiveness

In this section we present experiments and their results regarding the effectiveness of our modified version of the XACBench prototype tool and its XACML 3.0 to ASP translation functionality for analysing policy properties.

The experimental process is as follows:
First, we run our extended XACBench XACML3.0 to ASP translation functionality on a set of real world XACML 3.0 policies: PolicySetMedicalDemo_Dhouha, kmarket-sliver-policy, kmarket-gold-policy, kmarket-blue-policy, continue-a, xacml3-policyset-sli. The translation of each policy generates a set of ASP programs. Note that attribute values within attribute type domains of each translated Policy Set are created by the translator by parsing the input policy. Manual changes to values within attribute type domains, performed by an administrator for example, could lead to different results in property analysis. In the next step of the experimental process, we define and add a new ASP program(count_properties.asp) to each set of translations, which helps us track the number of occurrences of atoms that verify certain properties. Finally, we run the Clingo (clingo --stats=2) solver once for each set of ASP programs and it returns an answer set which contains atoms that describe various properties of the initial real-world policy.

4.2.1.1 Results

Real-World Policy	NR	CR	GR	SR	SRR	USR	URR
PolicySetMedicalDemo_Dhouha	6	0	0	0	0	2	2
continue-a	298	0	32	32	48	0	197
kmarket-gold-policy	3	0	2	0	1	2	2
kmarket-sliver-policy	5	0	4	0	2	2	2
kmarket-blue-policy	4	0	3	0	2	2	2
xacml3-policyset-sli	9	0	0	0	0	0	0

Figure 4.1: The number of: rules (NR), correlation anomalies between rules (CR), generalization anomalies between rules (GR), shadow anomalies between rules (SR), simple redundancy anomalies between rules (SRR), simple useless rules (USR), unreachable rules (URR).

Real-World Policy	NR	CP	GP	SP	SRP	USP	URP
PolicySetMedicalDemo_Dhouha	6	0	0	2	1	2	2
continue-a	298	0	0	0	1	0	165
kmarket-gold-policy	3	0	0	0	0	0	0
kmarket-sliver-policy	5	0	0	0	0	0	0
kmarket-blue-policy	4	0	0	0	0	0	0
xacml3-policyset-sli	9	0	0	0	0	0	0

Figure 4.2: The number of: rules (NR), correlation anomalies between policies (CP), generalization anomalies between policies (GP), shadow anomalies between policies (SP), simple redundancy anomalies between policies (SRP), useless policies (USP), unreachable policies (URP).

Real-World Policy	NR	INC	URPS	USPS	ISO	TR
PolicySetMedicalDemo_Dhouha	6	12	0	0	0	4
continue-a	298	0	0	0	3476	33
kmarket-gold-policy	3	0	0	0	0	2
kmarket-sliver-policy	5	0	0	0	0	2
kmarket-blue-policy	4	0	0	0	0	2
xacml3-policyset-sli	9	241	0	0	0	0

Figure 4.3: The number of: rules (NR), requests not matched by policy (INC), unreachable Policy Sets (URPS), useless Policy Set (USPS), isomorphisms (ISO), total redundancies (TR).

Figures 4.1, 4.2, and 4.3 demonstrate the number of occurrences of different properties of each one of the real-world policies. More specifically, Figure 4.3 shows numbers of occurrences of properties associated with the Policy Set element of XACML 3.0 (apart from TR which is associated with rules and policies), while figures 4.1 and 4.2 show properties which are associated with rule and policy elements of XACML 3.0 respectively.

In Figure 4.3 we see that xacml3-policyset-sli has 241 requests that could not be matched by the policy, which makes it incomplete. This incompleteness can lead to a variety of security problems. An example of a security threat due to incompleteness could be the following: an access-controlled system that by default permits indetermined requests paired with an attacker that purposefully creates requests which lead to neither permit nor deny decisions. In this case the attacker by default is permitted to perform malicious requests. The detection of unmatched requests can incentivise the administrator to either define new rules and policies that will lead the request to a deny or permit decision or adjust attribute domains so that such requests cannot be created. We can also see in Figure 4.3 that continue-a has many isomorphisms, which means that many pairs of Policy Sets within the policy match the same set off requests and also have the same affect. In other words, there are unneeded Policy Sets. An administrator can remove one of the Policy Sets from the isomorphic pair without altering the authorization scheme (decisions for requests) of the policy. Removing unneeded Policy Sets result in a smaller policy, which in turn leads to faster decision making and faster policy analysis itself.

Speedup of the analysis process can also be achieved by removing rules(policies) which are irrelevant, meaning that even if they were to be removed the policy's scheme would remain the same. Irrelevant rules(policies) are either simply redundant, totally redundant, shadowed by some other rule(policy), unreachable, or useless. In figures 4.1 and 4.2 we can see that continue-a has many unreachable rules and policies respectively. An administrator could remove these elements without affecting the policy's decision making.

4.2.2 Efficiency

In this section we present experiments and their results regarding the efficiency of our modified version of the XACBench prototype tool and its XACML 3.0 to ASP translation functionality in analysing policy properties.

The first experimental process is as follows:

We take a real world XACML 3.0 policy called PolicySetMedicalDemo_Dhouha and use the XACBench synthetic policy generator to create synthetic XACML 3.0 policies with varying number of rules (50, 100, 150, 200, 250, 300). Then we run our extended XACBench XACML3.0 to ASP translation functionality on all synthetic policies generated from PolicySetMedicalDemo_Dhouha, where each translation generates a set of ASP programs. For all sets of ASP programs generated we manually define a common set of attribute values for each attribute type domain, with the total number of requests being (15200). Finally, we run the Clingo (clingo --stats=2) solver once for each set of ASP programs. In this experiment we evaluate the efficiency of the tool based on the total time it took for the solver to generate the answer sets (CPU Time), and the number of ground rules generated as opposed to the number of rules in the XACML policy.

The second experimental process is as follows:
We take a real world XACML 3.0 policy called PolicySetMedicalDemo_Dhouha and use the XACBench synthetic policy generator to create a synthetic XACML 3.0 policy with 300 rules. Then we run our extended XACBench XACML3.0 to ASP translation functionality on the synthetic policy. This generates a set of ASP programs. Then step by step we change the attribute values within the domains of the set of ASP programs. At each step we change the number of values so that the total number of generated requests is 1120, 2080, 4160, 7904, 15200. In other words, we close to double the number of requests generated at each step. In between the steps, we run the Clingo (clingo --stats=2) solver on the set of ASP programs. In this experiment we evaluate the efficiency of the tool based on the total time it took for the solver to generate the answer sets (CPU Time), and the number of ground rules generated as opposed to the number of generated requests (attribute type domain size).

4.2.2.1 Results

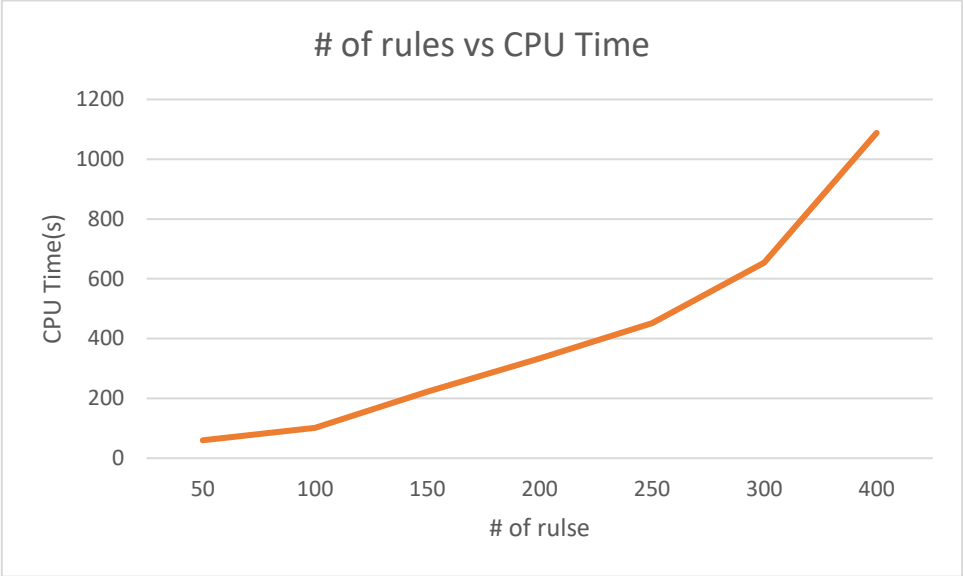


Figure 4.4: Run time affected by number of policy rules.

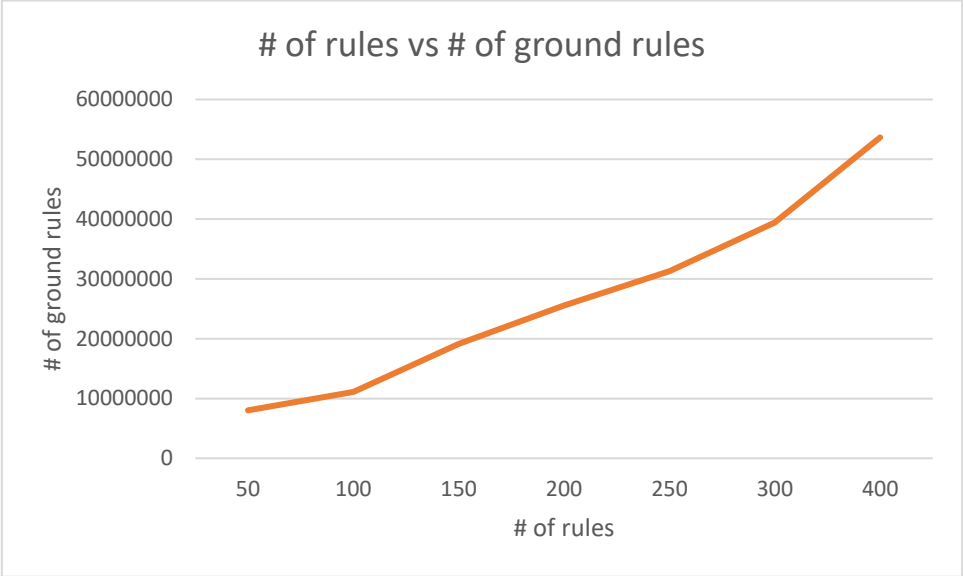


Figure 4.5: Number of ground rules affected by number of policy rules.

Figures 4.4 and 4.5, illustrate how the solving time and number of ground rules generated during solving is affected by the size of XACML 3.0 policy regarding the number of rules it contains. We can see that both the CPU Time (run time, solving time) and the number of generated ground rules grows exponentially compared to the number of rules. We must add that the largest policy (number of rules within it) we managed to run successfully was with 400 rules. We ran tests on policies with 450, 500, 600 rules and the solver did not manage to generate answer sets. This observation can be explained by the grounding process and its

exponential space complexity. Grounding P on a program D, which in our case are the ASP translation programs, leads to a propositional program P' whose size is exponential in the size of the fixed input database D [7]. Increasing the number of rules within a XACML 3.0 policy leads to a larger input database D.

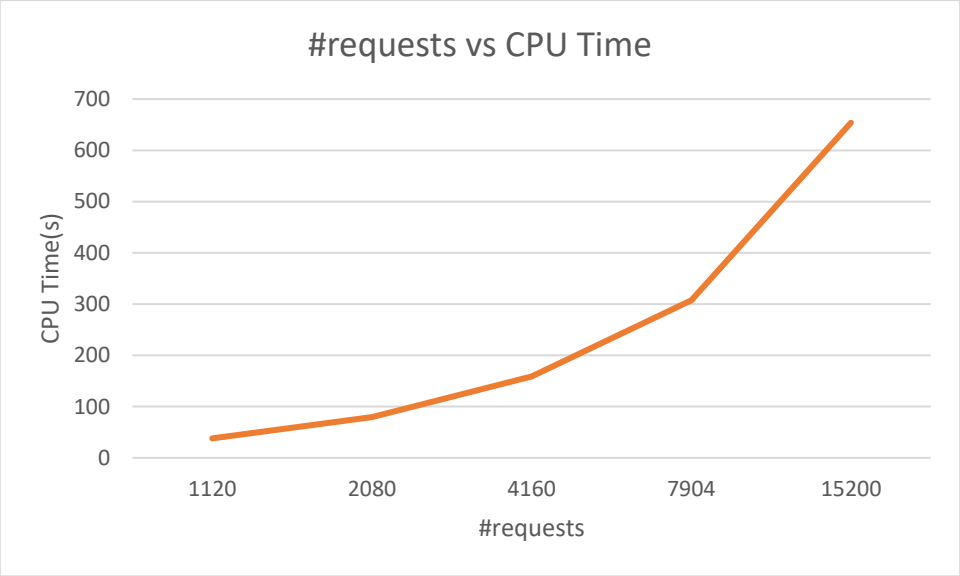


Figure 4.6: Run time affected by total number of generated requests (attribute type domain size).

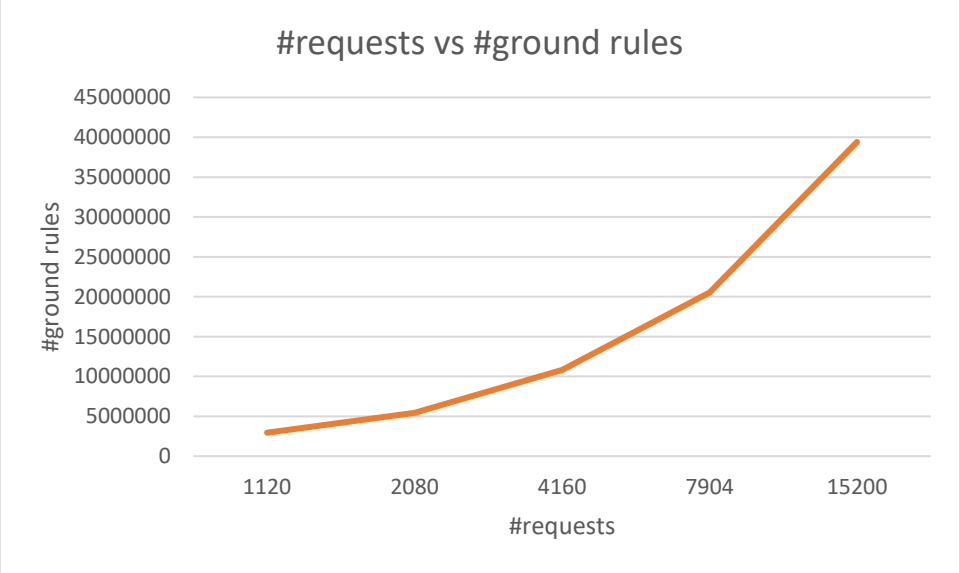


Figure 4.7: Number of ground rules affected by total number of generated requests (attribute type domain size).

Figures 4.6 and 4.7, illustrate how the CPU Time (run time, solving time) and the number of ground rules generated is affected by the total number of generated requests. We can see that

both the CPU Time (run time, solving time) and the number of generated ground rules grows exponentially compared to the total number of generated requests. This observation can again be explained by the grounding process and its exponential space complexity. Grounding P on a program D, which in our case are the ASP translation programs, leads to a propositional program P' whose size is exponential in the size of the fixed input database D. Increasing the domain size of a XACML 3.0 policy leads to a larger number of requests being generated which in turn leads to a larger input database D.

4.3 Efficiency of variation ASP program for query analysis

In this section we present experiments and their results regarding the efficiency our variation ASP programs and their query analysis functionality. The goal of the variation set of ASP programs was to make query analysis more efficient by reducing the arity (number of parameters) of atoms within the ASP programs which in turn would lead to faster grounding times.

The experimental process is as follows. We take a real world XACML 3.0 policy called PolicySetMedicalDemo_Dhouha and use the XACBench synthetic policy generation functionality to create synthetic XACML 3.0 policies with varying number of rules (50, 100, 150, 200, 250, 300). Then we run our extended XACBench XACML3.0 to ASP translation functionality on all synthetic policies generated from PolicySetMedicalDemo_Dhouha, where each translation generates a set of ASP programs. For all the synthetic policy translations we manually define a common set of attribute values for each attribute type domain, with the total number of requests being (11968). For each set of ASP programs we then define (Section 3.5) a variation ASP set of programs by manually making adjustments the ASP programs; main.pl, match_target_v3.asp, match_common.asp, out.asp. We make sure that the variation ASP set uses the request generator generate_one.asp instead of generate_all.asp. Finally, we evaluate query analysis efficiency, by running the Clingo (clingo --stats=2) solver once on the standard translation set and once on the variation set. Each pair of, synthetic policy translation, and its variation are compared based on the time it took to for the solver to find the solution (answer set), the number of generated rules, atoms, and bodies.

4.3.1 Results

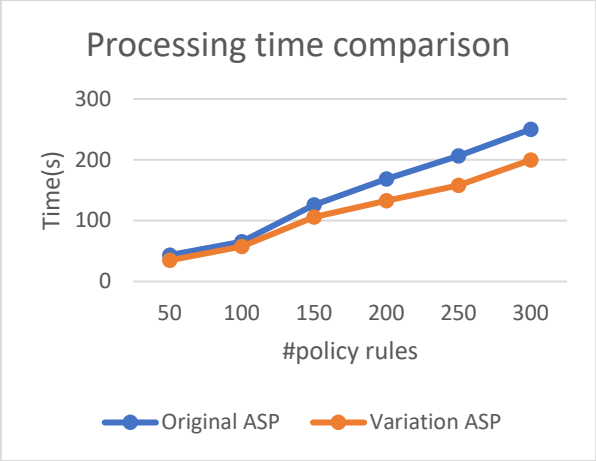


Figure 4.8: Processing time comparison.

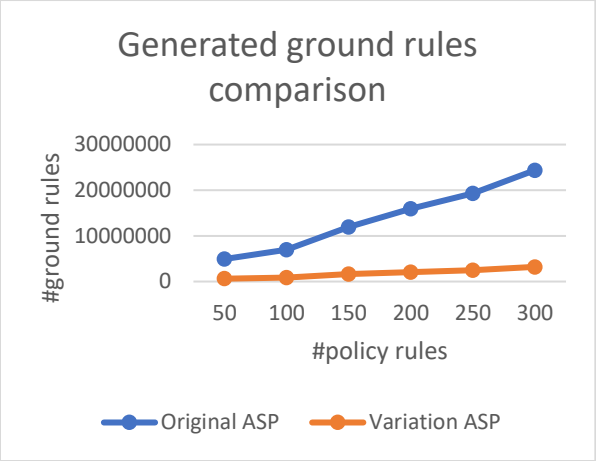


Figure 4.9: Number of ground rules comparison.

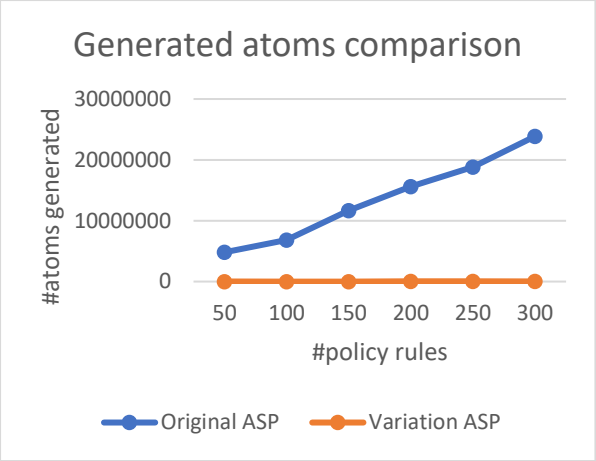


Figure 4.10: Number of atoms comparison.

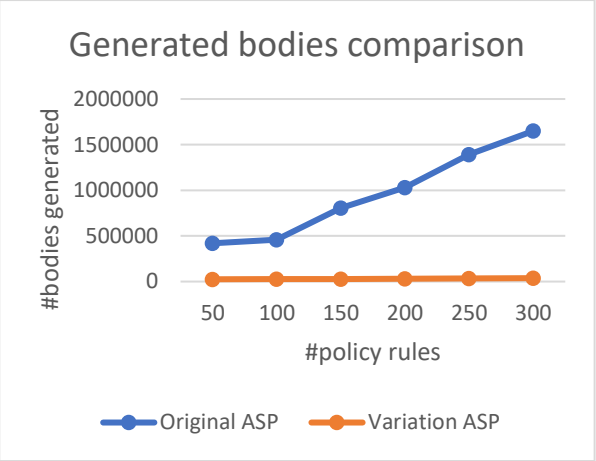


Figure 4.11: Number of bodies comparison.

Figure 4.8 shows the solver processing time (combination of grounding and solving time) of query analysis for the original ASP programs and the variation ASP programs where we can see that the variation version is quite faster (1.232 average speedup) than the original. We can also see in Figures 4.9, 4.10, and 4.11 that the number of generated ground rules, atoms, and bodies using the original set of ASP programs grows at a much faster rate compared to the variation set of ASP programs when the number of rules of the XACML 3.0 policy increases. This observation can be explained by the grounding process and its exponential space behaviour with the arity (number of parameters) of atoms being a big part of the exponent. Since the atoms within the variation ASP programs have a smaller number of parameters the grounder generates a much smaller number of ground rules, which in turn leads to a smaller number of atoms, bodies, and of course faster grounding times.

4.4 Hierarchy

In this section we present experiments and results regarding the hierarchy property we have added to the extended XACBench tool.

The experimental process is as follows:
We take a real world XACML 3.0 policy called PolicySetMedicalDemo_Dhouha.xml and use it as input for our extended XACML3.0 to ASP translation functionality. Once the translation is complete, a set of ASP programs is generated. Among those programs is hierarchy.asp, which contains rules based on the logic defined in Section 3.6. After the translation is completed, we manually add some values to the attribute type domains in the ASP program "generate_all.asp". Finally, we run the Clingo solver on the set of ASP programs and get an answer set which contains the following hierarchy related atoms:

top_hier(administrator,urnoasisnamestcxacml2_0subjectrole)
hier(administrator,clinical_researcher,urnoasisnamestcxacml2_0subjectrole)
hier(administrator,radiologist,urnoasisnamestcxacml2_0subjectrole)
hier(administrator,physician,urnoasisnamestcxacml2_0subjectrole)

top_hier(europe,urnoasisnamestcxacml2_0subjectlocation)
hier(europe,not_europe,urnoasisnamestcxacml2_0subjectlocation)

top_hier(val_de_grace,urnoasisnamestcxacml2_0subjecthospital)
hier(val_de_grace,not_val_de_grace,urnoasisnamestcxacml2_0subjecthospital)

top_hier(personal,urnoasisnamestcxacml1_0resourceresource_confidentiality)
hier(personal,not_personal,urnoasisnamestcxacml1_0resourceresource_confidentiality)

top_hier(medical_file,urnoasisnamestcxacml1_0resourceresource_id)
top_hier(radiography,urnoasisnamestcxacml1_0resourceresource_id)
top_hier(patient_record,urnoasisnamestcxacml1_0resourceresource_id)

top_hier(read,urnoasisnamestcxacml1_0actionaction_id)
top_hier(write,urnoasisnamestcxacml1_0actionaction_id)

From these atoms we build a more visually comprehensive representation of the hierarchies, using a graph for each attribute type [8]. The last parameter value of each atom indicates to which attribute type (graph) the atom belongs to.

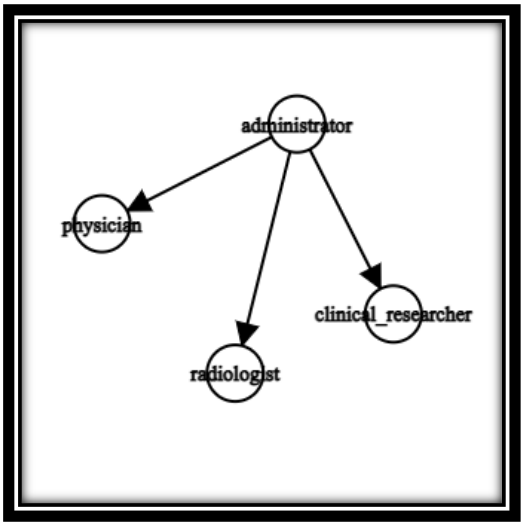


Figure 4.12: Role hierarchy graph.

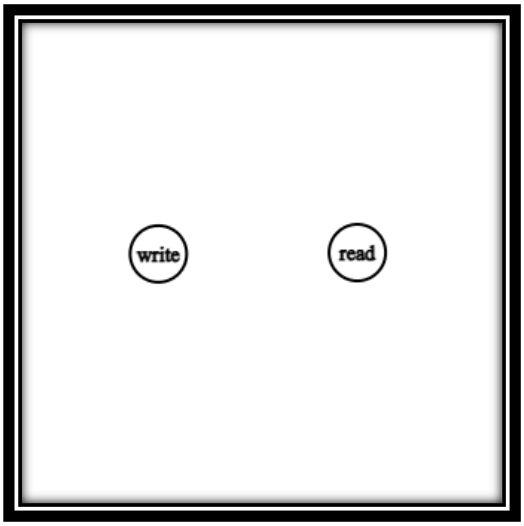


Figure 4.13: Action hierarchy graph.

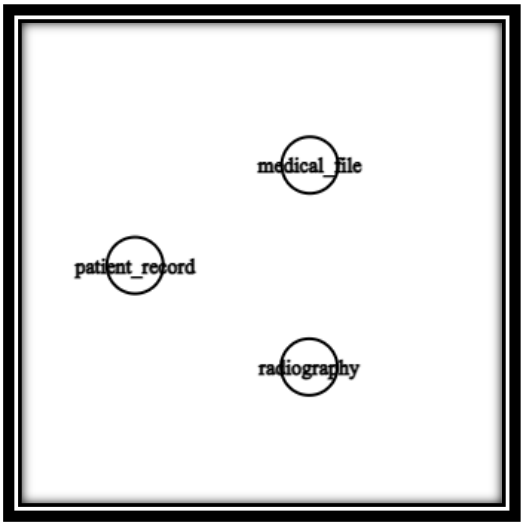


Figure 4.14: Resource hierarchy graph.

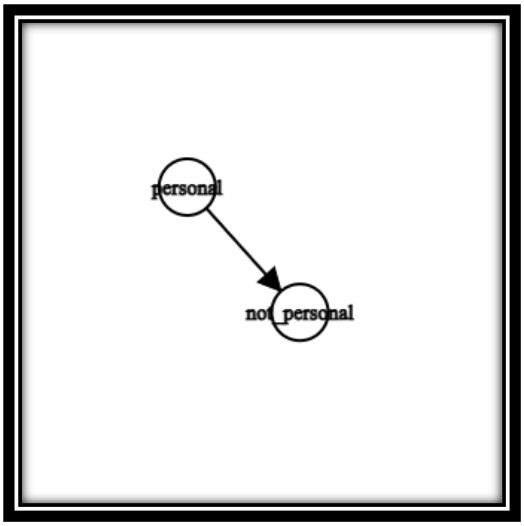


Figure 4.15: Confidentiality hierarchy graph.

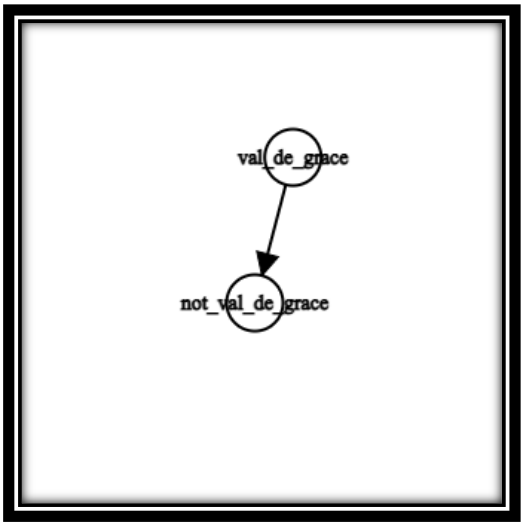


Figure 4.16: Hospital hierarchy graph.

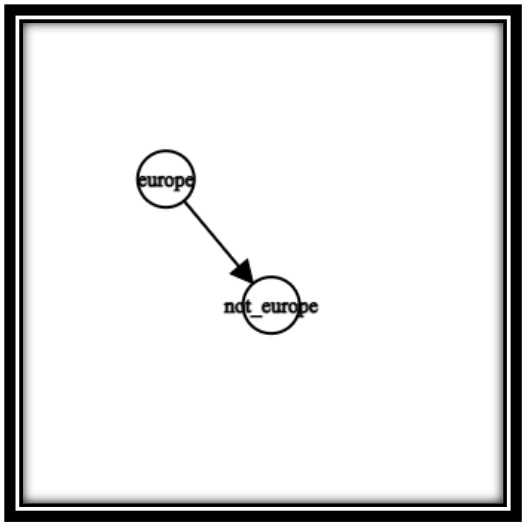


Figure 4.17: Location hierarchy graph.

In Figures 4.12, 4.13, 4.14, 4.15, 4.16, and 4.17 we can see the hierarchical graphs for the attribute types subjectrole, action_id, resource_id, resource_confidentiality, subjecthospital, and subjectlocation respectively. Nodes with no arrows pointed towards them are attribute values which are on top of the associated attribute type hierarchy. On the other hand, nodes that have arrows pointed towards them have a subset of the authority of the nodes that point towards them.

Chapter 5

Conclusions

5.1 Summary 92
5.2 Limitations 93
5.3 Future work 93

5.1 Summary

Our goal in this thesis was to study and extend a tool that would allow developers and analysers of web access policies to efficiently and effectively analyse XACML 3.0 policies. Our first objective was to modify an already existing prototype tool called XACBench so that it could accurately translate XACML 3.0 policies. By adjusting XACBench's java code and some of its predefined ASP programs we managed to create a tool capable of handling XACML 3.0s arbitrary attribute types. Given a XACML 3.0 policy as input, the modified tool creates an accurate translation set of ASP programs that can then be used by an ASP solver such as Clingo to effectively analyse the access policies as show in Section 4.2.1. As an overall showcase of our work regarding translation of XACML 3.0 policies to ASP programs we show two different translations of the real-world XACML 3.0 policy called "kmarket-gold-policy.xml". In Appendix C we have the first translation, which was generated using the original XACBench tool and in Appendix D we can see the second more accurate translation generated by our modified tool. Unfortunately, we did not manage to come to a satisfactory conclusion regarding the tool's efficiency in policy property analysis, since our experiments with translations of large XACML 3.0 policies (i.e., 500 rules) that were given to the Clingo solver could not generate answer sets in satisfactory time frames. We believe that this is caused by the grounding

process's exponential space complexity and the limited capabilities of our experimental environment. Our second objective was to create a variation set of ASP programs that would be more efficient in policy query analysis than the translation set of ASP programs generated by the modified XACBench tool. The variation set of ASP programs created had a reduced number of parameters in its atoms which we believed would lead to faster grounding and in turn to faster query analysis of XACML 3.0 policies. This variation did in fact lead to a considerable average speedup of 1.232. Lastly, we defined a new property for XACML 3.0 policy property analysis called hierarchy. More specifically, we created a policy dependent ASP program that contains rules capable of generating atoms which describe hierarchies of each attribute type found in access policies. Using a real-world policy, we managed to show how this property can give developers and analysers of access policies insight into the hierarchical authorization structure of different attribute type values within a policy.

5.2 Limitations

The experiments of our extended tool's policy property analysis and the results they yielded were largely affected, in a negative way, by the experimental environment and the computational power we had at hand. As we have mentioned before, the grounding size is exponential in the size of a given XACML policy, either in the number of rules or in the size of the domains of attribute types. This exponentiality together with the fact that the experiments were conducted on a single computing machine, as described in Section 4.1.1, obstructs us from coming to a satisfactory conclusion regarding the tool's efficiency in policy analysis.

5.3 Future work

For future testing and a more accurate evaluation of the tool's efficiency, we will need to create a more realistic experimental environment, where the computational power is analogous to the size and complexity of the access policies enforced by real-world authorization parties. Besides testing, we can look to add a new functionality that will give insight into how an undesired property can be resolved or avoided. More specifically, we could add to the tool new ASP programs which rules that can generate atoms which in turn can help guide developers and analysers to resolving different policy properties and anomalies. In the future we could try and improve our tool's efficiency for property analysis, in a similar way that we did for query

analysis, by defining an ASP variation syntax that will contain atoms with smaller number of parameters. This will be a more challenging task that will need some ASP logic workarounds because during property analysis we are forced to generate all possible request in a single answer set in order to verify various properties of the access policy.

References

[1] Rezvani, M., Rajaratnam, D., Ignjatovic, A., “Analyzing XACML policies using answer set programming”, 26 November 2018. [Online]. Available: https://doi.org/10.1007/s10207-018-0421-5.

[2] Organization for the Advancement of Structured Information Standards (OASIS), “eXtensible Access Control Markup Language (XACML) Version 3.0”, 22 January 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[3] Kaufmann, B., Leone, N., Perri, S., Schaub, T., “Grounding and Solving in Answer Set Programming”. AIMag 2016, p. 26. [Online]. Available: https://doi.org/10.1609/aimag.v37i3.2672.

[4] University of Potsdam, “clingo and gringo, Potassco, the Potsdam Answer Set Solving Collection”. [Online]. Available: https://potassco.org/clingo/.

[5] Lierler. Y., "Basics behind Answer Sets", August 2020, pp. 1-2. [Online]. Available: http://works.bepress.com/yuliya_lierler/71/.

[6] Lifschitz, V., “Answer Set Programming”, 5 April 2019, pp. 9-31.

[7] Dantsin, E., Eiter, T., Gottlob, G., Voronko, A., “Complexity and Expressive Power of Logic Programming”, February 1999, pp. 4-15. [Online]. Available: https://www.researchgate.net/publication/200034372_Complexity_and_Expressive_Power_of_Logic_Programming.

[8] CS Academy, “Graph Editor”. [Online]. Available: https://csacademy.com/app/graph_editor/.

[9] Ahmadi, S., Nassiri, M., Rezvani, M., “XACBench”. [Online].
Available: https://github.com/nassirim/xacBench.

[10] E. S. Al-Shaer and H. H. Hamed, "Discovery of policy anomalies in distributed firewalls”, 2004. [Online].
Available: https://ieeexplore.ieee.org/document/1354680.

[11] Leone, N., Ricca, F., “Answer Set Programming: A Tour from the Basics to Advanced Development”, 2005. [Online].
Availabe:
https://www.researchgate.net/publication/300646285_Answer_Set_Programming_A_Tour_from_the_Basics_to_Advanced_Development_Tools_and_Industrial_Applications.

[12] Genesereth, M., Vinay K. Chaudhri, “Introduction to Logic Programming: (Synthesis Lectures on Artificial Intelligence and Machine Learning)”, February 2020. [Online].
Availabe:
https://www.researchgate.net/publication/339156051_Introduction_to_Logic_Programming/stats.

Appendix A

Real-world XACML 3.0 policy “kmarket-gold-policy”

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable"
    PolicySetId="RPSlist" Version="1.0">
    <Target />
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicyId="KmarketGoldPolicy" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides" Version="1.0">
   <Target>
      <AnyOf>
         <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">gold
               </AttributeValue>
               <AttributeDesignator AttributeId="http://kmarket.com/id/role"
               Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
               DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
            </Match>
         </AllOf>
      </AnyOf>
   </Target>
   <Rule Effect="Deny" RuleId="total-amount">
      <Condition>
         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
               <AttributeDesignator AttributeId="http://kmarket.com/id/totalAmount"
               Category="http://kmarket.com/category"
               DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">1000</AttributeValue>
         </Apply>
      </Condition>
    <AdviceExpressions>
    <AdviceExpression AdviceId="deny-liquor-medicine-advice" AppliesTo="Deny">
    <AttributeAssignmentExpression AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">You are not allowed to do more
    than $1000 purchase from KMarket on-line trading system</AttributeValue>
    </AttributeAssignmentExpression>
    </AdviceExpression>
    </AdviceExpressions>
   </Rule>
   <Rule Effect="Deny" RuleId="max liquor amount">
   <Target>
      <AnyOf>
         <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Liquor</AttributeValue>
               <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
               Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
               DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
            </Match>
         </AllOf>
      </AnyOf>
   </Target>
      <Condition>
         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
               <AttributeDesignator AttributeId="http://kmarket.com/id/amount"
               Category="http://kmarket.com/category"
               DataType="http://www.w3.org/2001/XMLSchema#integer" MustBePresent="true"/>
            </Apply>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
         </Apply>
      </Condition>
    <AdviceExpressions>
    <AdviceExpression AdviceId="max-drink-amount-advice" AppliesTo="Deny">
    <AttributeAssignmentExpression AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">You are not allowed to buy more
    tha 10 Liquor from KMarket on-line trading system</AttributeValue>
    </AttributeAssignmentExpression>
    </AdviceExpression>
    </AdviceExpressions>
   </Rule>
    <Rule RuleId="permit-rule" Effect="Permit"/>
</Policy>
</PolicySet>
```

Appendix B

Request for “kmarket-gold-policy”

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    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"
    ReturnPolicyIdList="false" CombinedDecision="false">
    <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
        <Attribute IncludeInResult="false"
            AttributeId="http://kmarket.com/id/role">
            <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">gold</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="http://kmarket.com/category">
        <Attribute IncludeInResult="false"
            AttributeId="http://kmarket.com/id/totalAmount">
            <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">550</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
        <Attribute IncludeInResult="false"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
            <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">Liquor</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="http://kmarket.com/category">
        <Attribute IncludeInResult="false"
            AttributeId="http://kmarket.com/id/amount">
            <AttributeValue
            DataType="http://kmarket.com/id/amount">20</AttributeValue>
        </Attribute>
    </Attributes>
</Request>
```

Appendix C

XACBench translation of “kmarket-gold-policy”

```
effect(permit;deny;indeterminate).
comb_alg(deny_overrides;permit_overrides;first_applicable;only_one_applicable).
comb_alg(do;po;fa;ooa).
bool_expr(true).

target(r1).
condition(r1, true).
rule(r1, 1, p1, deny).

anyof(r2, any2, any, liquor).

target(r2) :- anyof(r2 ,any2, _, _).
condition(r2, true).
rule(r2, 2, p1, deny).

target(r3).
condition(r3, true).
rule(r3, 3, p1, permit).

anyof(p1, any2, gold, any).

target(p1) :- anyof(p1 ,any2, _, _).
policy(p1, 1, ps0, deny_overrides).

target(ps0).
policy_set(ps0, 0, ps0, first_applicable).
```

Appendix D

Modified XACBench translation of “kmarket-gold-policy”

```
effect(permit;deny;indeterminate).
comb_alg(deny_overrides;permit_overrides;first_applicable;only_one_applicable).
comb_alg(do;po;fa;ooa).
bool_expr(true).

target(r1).
condition(r1, Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id):-
request( Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id),
Http__kmarket_com_id_totalamount>1000.

rule(r1, 1, p1, deny).

anyof(r2, any2, any, any, any, liquor).

condition(r2, Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id):-
request( Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id),
Http__kmarket_com_id_amount>10.

rule(r2, 2, p1, deny).

target(r3).
condition(r3, Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id):-
request( Http__kmarket_com_id_amount, Http__kmarket_com_id_role,
Http__kmarket_com_id_totalamount, Urnoasisnamestcxacml1_0resourceresource_id).

rule(r3, 3, p1, permit).

anyof(p1, any2, any, gold, any, any).

policy(p1, 1, ps0, deny_overrides).

target(ps0).
policy_set(ps0, 0, ps0, first_applicable).
```