The Conflict Notion and its Static Detection: a Formal Survey

Jean-Claude Royer

The Conflict Notion and its Static Detection: a Formal Survey

Jean-Claude Royer

Jean-Claude.Royer@imt-atlantique.fr

July 1, 2019

Abstract

The notion of policy is widely used to enable a flexible control of many systems: access control, privacy, accountability, data base, service, contract, network configuration, and so on. One important feature is to be able to check these policies against contradictions before the enforcement step. This is the problem of the conflict detection which can be done at different steps and with different approaches. This paper presents a review of the principles for conflict detection in related security policy languages. The policy languages, the notions of conflict and the means to detect conflicts are various, hence it is difficult to compare the different principles. We propose an analysis and a comparison of the five static detection principles we found in reviewing more than forty papers of the literature. To make the comparison easier we develop a logical model with four syntactic types of systems covering most of the literature examples. We provide a semantic classification of the conflict notions and thus, we are able to relate the detection principles, the syntactic types and the semantic classification. Our comparison shows the exact link between logical consistency and the conflict notions, and that some detection principles are subject to weaknesses if not used with the right conditions.

Keywords: Security Policy, Conflict, Chaining, Static Detection, Undefined Request

1 Introduction

Policies are common in computer science, they govern many systems, mainly security ones (firewalls, access controls, privacy, ...), accountable systems, data bases, telecommunication or information systems, internet services, system configuration, QoS contracts, and so on. Policies are behavioral rules regulating a system and providing flexibility and evolution of the controls. Writing such policies raise similar challenges than programming and several authors argue for a software engineering approach to the management of policies [1, 18, 27, 14, 62, 54, 50]. Often a declarative approach is suggested, it provides: abstraction, conciseness, and captures the business information without the burden of operational details. That is, it expresses the business logic abstracting from the control flow. This is similar to the benefits of logic programming and this suggests that a logical framework is often a good mean to understand and to check these systems.

One important problem is the detection of conflicts in policies [11, 71, 14, 28, 3]. A conflict is a situation where two or more contradictory results are raised from the same request or context. Most authors refer to the term "conflict", but others use inconsistency, several mix both and few talk about contradiction or incompatibility. There are many work about this issue, see the previous surveys [16, 22, 11, 59, 28, 4]. We focus here on detection of conflicts in a fixed set of policies, these policies can be of different nature, only permission, containing actions, complex effects or even changing over time. Languages are numerous, the notions of conflict are various and the detection techniques diverse. This is also due to the fact that reasons to look for conflicts are different. Some specifiers eliminate unenforceable systems, some wants to avoid some bugs, while others only discard ambiguous requests or non determinism in the policy triggering. In the past years, some related work argues that static detection is generally not possible and they suggest dynamic detection often with a conflict resolution. Dynamic approaches have been criticized [37, 35, 66, 23, 61, 62] and we will mainly discuss and compare static approaches. We can now found a set of work proposing a policy language and a static conflict detection mechanism, for instance [1, 18, 27, 53, 2, 14, 71, 49, 62, 66, 70, 67, 72, 5, 10, 20, 65, 61] among more than thirty. Analyzing the recent related work (since 2008) we extract four types of languages and five principles for the static detection of conflicts.

This analysis raises several questions about the nature of a conflict and the principles and conditions to detect them. Detecting conflicts by checking pairs of policy is popular in various contexts, but only few work discusses its weaknesses [55, 43, 61]. Looking for conflicts between pair of rules is not sufficient to ensure the absence of conflict in a rule system. Sometime existing detection mechanisms do not find a conflict while the system is inconsistent and then not enforceable at all. Furthermore, there are other problems which are not conflicts in the classic sense but they share a common behaviour and they should be analyzed.

Our contributions in this paper are: *i)* a comprehensive review of the main recent work in conflict detection, *ii)* a formalization of the five static detection principles coping with four kinds of systems and three semantic notions of conflicts in policy systems, and *iii)* a formal analysis ensuring a correct conflict detection and explaining few weaknesses we observed in related work.

The structure of this paper is as follows. We present our motivations and background in Section 2. Related work is described in Section 3, an informal analysis is exposed collecting the main remarks regarding the detection time, the conflict notions, the syntax of systems and the static detection principles. Section 4 is devoted to our formalization of the policy systems and the detection principles, a comparative map is drawn, and weaknesses are discussed. A final section summarizes our main findings.

2 Motivations and Background

Often a policy system uses rules with a premise and a conclusion, Listing 1 shows a simple example. Variables `X` and `Y` are implicitly universally quantified and `DENY` is the negation of `PERMIT`. Comparing the two rules one can find a conflict, a situation where entering is both permitted and denied.

Listing 1: A simple policy example

```
password(X, Y) => PERMIT enter(X) AND
technician(X)  => DENY enter(X)
```

In Listing 2 we have two logically equivalent systems to Listing 1. With the second version, comparing two rules is not relevant to raise a problem. The last line of the third version makes explicit that a correct implementation should throw away cases which lead to an inconsistency.

Listing 2: Two equivalent systems

```
// 2)
(NOT password(X, Y) AND NOT technician(X)) OR
(NOT password(X, Y) AND DENY enter(X)) OR
(NOT technician(X) AND PERMIT enter(X))
// 3)
(password(X, Y) AND NOT technician(X))
                        => PERMIT enter(X) AND
(NOT password(X, Y) AND technician(X))
                        => DENY enter(X) AND
(password(X, Y) AND technician(X)) => False
```

In this section we will make more precise our scope, our hypotheses on the policy systems and the logical tool we use to compare the different approaches.

2.1 The Conflict Notion

The policy conflict problem is formulated in the literature as: "is there an incompatibility in the policies of my system", "a logical inconsistency in my system", "a contradiction between the conclusion of two rules", or "an overlapping in my policies". The policy conflict problem is an important issue [11, 71, 66, 28, 3] which should be analyzed before any other policy analysis, enforcement or request evaluation. Of course, features of the policy language and conflict concerns are often different thus conflict detection should be different.

We are in fact interested in a general notion of conflict, not domain dependent as in some work like [53, 71]. For example: An action which is both permitted and denied, an obligation without permission, an obligation and its prohibition, etc. We focus on what we should call "true" or *real conflicts* in the sense that they denote a system inconsistency, an implementation bug, or a request leading to a failure. Redundancy is another problem entailing performances and readability but not the system behaviour. These conflicts are also different than incompleteness problems, or misconfigurations which are rather holes or irregularities in the policy system. We think that all kinds of real conflict can be managed in a uniform way provided that we have the good language and the good tool support. One of our contribution is to provide formal definitions which cover these notions of conflict. As quoted in [61] there are three steps in conflict management: detection, localization, and resolution. We here only focus on detection but some of our related work study more, often localization and sometime resolution are mixed with detection. We also do not discuss and compare the complexity and the performance of the different implementations.

2.2 Policy Systems

Our context is mainly security policy and privacy but the policy conflict problem is also relevant in several other areas like data base management, firewall, networks, and quality of services contracts. Basically, an access control system established action permissions for some subjects on some resources. Policies can be stratified or not, they need complex conditions, and negation can play a role. But often we found also obligations to fulfill and subjects can be organized with roles or other kinds of grouping. Often we have attributes for resources and subjects, hierarchies are often a convenient mean to simplify and organized policies. Privacy concerns [7, 17, 20] add some specific features: notification, purpose, explicit consent, data retention but also data dis-

closure and processing locations. Usage access control mixes all these possibilities [67]. Abstractly, these policies or rules compound a system which upon a request provides a reply which can be an authorization (positive or negative), a mandatory actions, a new role, and so on. A policy set is generally viewed as a conjunction of policies, [55, 66] comment this choice. This is also the assumption we made here.

2.3 The Logical Framework

To understand and compare the numerous approaches, we need to abstract some of the language details. We will look for the founding principle behind the various algorithms, generally it is expressed informally, sometime with a formal definition. We will consider a logical framework to compare these principles because it easily abstracts from implementation details and this way is often used by several existing proposals. Another benefit of a logical approach is to be flexible regarding the composition of policies, conjunction and disjunction can be freely supported.

There are several logical frameworks which have similar properties and can be used to represent or interpret policies. These logical frameworks are: $\mathbb{PROP}$ (Propositional logic), $\mathbb{FOL}$ (First-Order Logic), and $\mathbb{FOTL}$ (First-Order Linear Temporal Logic). Some others are sometime used like $\mathbb{LTL}$ (Linear Temporal Logic, or similar variants like Propositional Linear Temporal Logic), Propositional Interval Logic (and variants), and description logic (OWL-DL, SWRL).

We will shortly describe $\mathbb{FOTL}$ as it subsumes the different logic we observe in our state of the art. $\mathbb{FOTL}$ formulas are built from variables V, functions F and predicates P with a fixed arity. Terms are built recursively over variables and functions, and if P_i is a predicate symbol, (e_j) terms then $P_i(e_j)$ is an atom (A). Syntactically and semantically $\mathbb{FOTL}$ subsumes $\mathbb{PROP}$, $\mathbb{FOL}$ and $\mathbb{LTL}$. $\mathbb{FOTL}$ is not a decidable logic but there are several fragments with decidable properties. A detailed presentation with a decision procedure can be found in [48], more practical explanations are available in [24].

Listing 3: First-Order linear Temporal Logic

```
F ::= A // Atomic formulae
   // Boolean operators
   + (NOT F) +  (F AND F) + (F OR F) + (F => F)
   // Quantifiers
   + (FORALL V F) + (EXISTS V F)
   // Linear temporal operators
   + (ALWAYS F) + (SOMETIME F) + (NEXT F) + (F UNTIL F
     )
```

A model for a $\mathbb{FOTL}$ formula is a linear temporal structure which associates to each integer a first-order structure, compound of a non-empty domain of values and an interpretation of functions and predicates. A decision procedure is an algorithm which allows to derive theorems using logical entailment. Obviously, the definitions of model and decision procedure are specific to each logic but other concepts can be abstracted. We recall here some classic concepts which are common to $\mathbb{PROP}$, $\mathbb{FOL}$ and $\mathbb{FOTL}$. A formula is *satisfiable* if and only if it has a model, while it is *valid* if it is satisfiable for all models, or its negation is unsatisfiable. Given a formula its validity (noted $\models$) can be asserted using the semantic model construction or can be proved using the syntactic decision procedure (noted $\vdash$). A logic is *complete* if and only if these two notions are equivalent. A formula is *inconsistent* (not satisfiable, or unsatisfiable) if it exists Φ a formula and we can derive Φ and $\neg\Phi$. The completeness property exists for classic logic: $\mathbb{PROP}$, $\mathbb{LTL}$, and $\mathbb{FOL}$. For $\mathbb{FOTL}$ with the monodic constraint, [19] has demonstrated the completeness of the temporal resolution process.

3 State of the Art

There are many references related to the problem of conflict detection in security policies. We comment here only the recent references which are dedicated to conflict detection and mainly published since 2008. We collect article references from browsing internet and several dedicated libraries comprising: IEEE Explore Digital Library, ACM Digital Library, DPLB, and The Collection of Computer Science Bibliographies. We classify these approaches in *surveys, dynamic, testing, static, and mixed approaches*. However, we do not consider few papers which are without sufficient enough information or have less than six pages. We focus on security, privacy, access control and related work but we exclude papers specific to firewalls. The main reason is that firewall languages are limited to permit and deny rules, or a finite set of incompatible literals as effects. The conflict detection principle is to use a pairwise approach. This case is covered by our classification, namely Type 1, we will discuss it later. In fact this bulk of work is much more interested in efficient solutions, a finer classification of anomalies or advanced features like automatic resolution which are out of our scope. Thus we exclude the following references [26, 71, 33, 68, 31, 15, 29], from our analysis.

We explored the previous *surveys* on this subject: [16] mainly for historical reasons, [22] is a general survey of static and dynamic conflict detection and resolution, [11] provides an overview of the state of the art in policy conflict detection and resolution, [59] studies five policy frameworks and [28] is a general survey about policy management in network systems. [4] is the most recent and comprehensive survey of access control policy systems and their validation mechanisms. The authors reviewed 26 papers from 2005 to 2014 and only 23 with a conflict management feature. The only mentioned principle is the pairwise rule comparison, called Level 1 in our analysis. None of the previous surveys collects and formally compares the main conflict detection principles.

Apart the surveys, we reviewed 46 references, see Table 1. Some approaches suggest a *dynamic* detection and generally an automatic resolution. The principle is to analyze, at runtime, each request and to look for conflicting results. In case of conflict a combining algorithm, a meta-policy, or a priority are used to select one reply to the request. First of all is the XACML engine [58] and [52, 38, 23] rely on this way. Few papers [36, 51, 69, 40] propose a formal way to verify XACML by translating it in a more formal language and then use a verification tool support. A second method, *testing* is to statically generate a set of requests and to test if a conflict occurs using the request evaluation process [37, 47]. The *static* approach is to rely on an algorithm, at design time, to statically check for conflicts and without an explicit generation and evaluation of the requests. These approaches are numerous, they generally rely on dedicated algorithms or the use of a logical prover. 34 references are collected in Table 1. Note that we only refer to [62] as it extends the previous work in [63], [45] as it overlaps similar references from 2013 and 2015 and [32] rather than the older one from 2011. Finally few proposals [56, 64] consider a *mixed* way: The conflict detection is mainly static and dynamic for the remaining conflict cases.

3.1 Detection Time

Table 1 summarizes the classification our 50 references from 2008. We recall the main criticisms

Table 1: Detection time classification.

	Survey	Dynamic	Testing	Static	Mixed
2008	[11, 59]	[36]		[1, 18, 27, 53]	
2009			[37]	[2, 14, 41, 35]	
2010			[47]	[42, 43, 49, 57, 55, 66, 70]	
2011		[52]		[60, 67]	
2012	[28]			[9, 54, 72, 5, 74]	
2013		[58, 38]		[3, 10, 20, 32]	
2014		[23]		[73, 8]	[56]
2015	[4]	[51, 69]		[39]	[64]
2016				[65, 61]	
2017		[40]		[21, 45, 62]	
Total	4	8	2	34	2

to dynamic approaches, [2, 37, 66, 72, 52, 32, 23, 61, 62] comment also these problems. Using sequential ordering, combining algorithms or decision rules are meta-policies which are outside the core security. These are often ad-hoc solutions to avoid conflicts, they are not resulting from security requirements and trouble the policy author. [6] also criticized this point as it forces consistency but prohibits local reasoning. Local reasoning is essential to enable modularity which is critical to maintain large sets of policies. The dynamic detection also increases the redundancies, the conflict cases and it is not scalable [59, 5]. Scalability is an important issue because of several policy sets, written by different policy officers. An automatic management is needed, but it is not possible because merging several sets of rules will add new conflicts and combining algorithms are not closed by composition. For example, the set of combining algorithms was enlarged from XACML V2 to its current V3 but as quoted by [22, 46, 66, 23] this is not sufficient. Indeed, the dynamic resolution has a fundamental weakness: the detection cannot distinguishes between a conflict coming from an error, introduced during the design and development steps, from a real contradiction in the requirements. Thus, in general, the right solution cannot be determined a priory in an automatic way. The policy officer should solve it and of course he needs tools to detect, localize and analyze the conflict.

The testing solution has generally two problems: *i)* the cost of the request generation and evaluation, and *ii)* the completeness or coverage of the request set.

One strong advantage of the static detection compared to the use of meta-policies is that in case of merging it is an automatic approach, it does not need a combining algorithm and it does not add runtime overhead. Another critical benefit is that the conflict is immediately reported to the privacy or security officer, at design time, and resolved by him which is the only one who can reasonably find a correct solution. However,

it requires a correct algorithm and as efficient as possible. We will focus, in the sequel, on the static detection of conflicts. We are interested in comparing the principles and the accuracy of the different proposals, algorithms complexity and performances are not discussed.

Mixed approaches as they allow both static and dynamic detection are surely the best approaches. A static approach could be always completed by a dynamic one checking for the residual cases which are not statically detectable. We do not further consider mixed approaches in our analysis, it does not change our detection principles as well as our conclusions.

3.2 Real Conflict

As previously explained in Section 2.1 we are concerned here with conflicts leading to real problems in the implementation of a system or in the request evaluation process. Some authors consider that conflicts are domains dependent ([53, 71]) but most of the papers thinks that a conflict is a general problem linked to opposite conclusions in permissions, roles, obligations and so on. The notion of functional conflicts (as opposed to logical conflict in [3]) enters in this category. Domain dependent conflicts can be managed by specifying the conflict property and the use of a verification mechanism, a good example is [53]. Another alternative, used by approaches based on verification like [27, 10, 8, 61] is to consider them as logical conflicts by making explicit all the context conditions and specifying the contradiction cases. In [14] modality conflicts are defined as "the joint authorization and denial of a request to perform some action, or the presence of an obligation to act without the permissions necessary for its fulfillment", from [2, 41, 61] we add obligation and interdiction for an action, while [20] discusses conflict in case of data disclosure, and [67] mainly considers the purpose conflicts. [5] enriches it with both a negative and positive role. In several of the previous work ([1, 2, 49, 42]) the notion of conflict is confused with the notion of inconsistency.

3.3 Type of System

Policy sets are sometime organized in two levels: rules and policies as in XACML. The basic elements of a policy system are called policy, rule, clause, sentence, etc. Features but also syntactic presentation of policies are varying. A rule is often an `IF THEN` control structure, an Event-Action-Condition construction, a tuple of several atoms with a qualifier, and many others used in policy languages are considered as rules here. The syntax of rules is an important issue and critical aspects are variables, negation without restriction and specific modal operators. Some frameworks use simple rules with only positive atoms in conditions while other rely on $\mathbb{FOL}$ implications. This makes a great difference in the ability to decide for conflicting situations. We saw some simple rules, often presented as tuples [18, 35, 49]. We observe explicit rules in the following papers [1, 27, 62, 70, 20]. One point to note is that the syntax of the conclusions of rules makes a difference from the expressiveness point of view. In simple access control we only need permit and deny as conclusions but if the language allows obligations it requires actions in conclusions. Conjunctions are common in policies, this is not true for disjunction, and sometime the negation is restricted, for instance on obligations. Of course introducing first-order variables increases the expressiveness. Often rules have universally and implicitly quantified variables. One further point to note is the

chaining of rules, often rules are not allowed to call other rules, in other words the request evaluation triggers only one rule to get the reply. But in some cases, [27, 14, 9], the deduction is more complex and chains several rules. It is required if we have permissions or obligations depending from other permissions or obligations like in complex protocols [27, 2, 61]. Sentences or logical formulae are also used in [41, 8, 61]. The precise syntax of the requests submitted to the policy engine is also an important aspect. Sometimes it is implicitly described as a tuple of valued attributes containing a subject, a resource and an action, but there are many other variations. Only few work precisely describes the set of requests and the constraints on the language to write rule conditions and conclusions.

3.4 Static Detection Principles

We found five principles to check for conflicts in a policy system.

1. A conflict occurs if two rules with overlapping conditions conclude with incompatible effects like permit and deny (15 references: [1, 18, 2, 57, 66, 60, 5, 54, 74, 32, 73, 39, 65, 62, 21]). Most of the references calls it a conflict but [62] formally defines the term inconsistency.

2. A conflict is a contradiction between the conclusions of two rules with overlapping conditions (4 references: [70, 67, 20, 3]).

3. A conflict is a contradiction between the conclusions of several rules with overlapping conditions. It was proposed and implemented in two references [55, 43].

4. A conflict occurs when a request has contradictory replies (2 references: [49, 72]). This principle is also used by the dynamic and the testing approaches.

5. A conflict is a logical inconsistency, that is Φ and $\neg\Phi$ are both valid derivations of the proof system (9 references: [53, 27, 14, 35, 41, 9, 10, 8, 61]).

Among the 34 static references, there are two exceptions: [42, 45] which do not appear in this classification because of lack of details, the conflict detection is not precisely described. Detecting conditions overlapping use different techniques, all are instances of the satisfiability of the conjunction of two conditions. Principle 1 uses satisfiability of conditions and incompatibility in rule effects. It collects references with only permit and deny as conclusions ([1, 2, 66, 60, 32, 73, 39, 65, 62, 21]) and others have a finite set of literals. Principle 2 adds general contradiction in conclusions. Principle 3 considers a finite (not fixed) set of conflicting rules. Between Principle 2 and Principle 3 we may have intermediate cases with a fixed number of rules (3, 4, or more). But it does not carry interesting properties, it was experimented by [43]. Principle 4 is based on request evaluation and is informally described as a set of requests leading to contradictory replies. In the testing approach of [47] the Level 4 is called an incompatibility while a conflict is defined as: any request has contradictory replies. Principle 5 is unsatisfiability or logical inconsistency, indeed we group approaches dealing with satisfiability checking ([27, 35, 61]) and those proving some consistency properties, like `NOT (PERMIT AND DENY)`, or specific domain properties ([53, 14, 41, 9, 10, 8]). These approaches use manual proof or derivation tools and are able to check logical consistency. One can further distinguish between the use of bounded satisfiability or an automated logical prover, but we do not

go into these details here. To get a more precise comparison of the above principles we need a more formal analysis which is the bulk of the next section.

4 Formal Analysis

To make precise our comparison we will consider a single logical framework with sentences or rules. We do not consider implicit rules (like close-world, what is not explicitly permitted is prohibited, etc), the policy system expresses syntactically all its behaviours with logical sentences. We provide here a light formal model in the sense that it relies on Boolean laws and basic set theory. We use the $\mathbb{FOTL}$ syntax for policy examples and classic formal notations to reason over the policy sets. Our analysis and results are valid in any complete logic extending propositional logic.

4.1 Policy System and Request

Our intuition is: A request defines a set of models and then in conjunction with a set of logical sentences, representing the policy, the deduction mechanism can prove (or not) an expected conclusion. It is important to define the notion of conclusion, which are the *replies*. Our basic policy elements, sentences, rules, requests and replies are according to the $\mathbb{FOTL}$ grammar, that is taken from the language $\mathcal{L}(\mathbb{FOTL})$. In the following, i, j, k, n are natural numbers and I, J, K are non empty finite sets of natural numbers included in $\{1 \,..\, n\}$. A policy system is set of logical sentences R, it is a logical conjunction of policies $R = \wedge_{1 \leq i \leq n}\, r_i$. R will be called a policy system, or system. When it is not confusing, we will consider R as a set of policies. A request will be logically represented by a satisfiable sentence noted req. A reply rep will be also a logical sentence. The deduction (or derivation) process will represent the query and reply mechanism, that is: $R,\ req \vdash rep$ is valid. It will be denoted by the validity of the natural implication $(R \wedge req) => rep$ or, equivalently, by the unsatisfiability of $(R \ \wedge \ req \ \wedge \neg rep)$.

Definition 4.1 (Undefined request)**.** *Let R, a satisfiable request req is undefined if and only if $req \wedge R$ is unsatisfiable.*

Otherwise the request is *defined* and that means that $R \wedge req$ is satisfiable.

4.2 Policy Types

A sentence is a logical $\mathbb{FOTL}$ expression while a rule will be a pair of logical expressions noted $A => B$ with the implication operator. A *rule system* is a set of n rules noted $\wedge_{1 \leq i \leq n}(cond_i => conc_i)$. From our analysis in Section 3.3 we observe four types of policy system. Type 1 covers pure access control languages, but also discrete roles, location and obligation features. Type 2 allows complex conclusions but without composition of rules, that is a conclusion of a rule cannot be used in a deduction with another rule. Type 3 allows more complex deductions by chaining the rules, thus enabling complex dependencies in policies. Type 4 has no restriction this is $\mathbb{FOTL}$. We introduce a notion of *request chaining* to distinguish between Type 1, 2 and Type 3. With Type 1 and 2 a reply results from the parallel deductions of some rules while Type 3 allows parallel and sequential deductions.

Definition 4.2 (Request chaining)**.** *Let a rule system R, $n \geq 2$, req a request has the chaining property if and only if*

$\exists I$, $(req \wedge_{i \in I} \ conc_i)$ *is satisfiable,* $req \wedge_{j \notin I} \neg cond_j$ *is satisfiable, and* $((req \wedge_{i \in I} \ conc_i) => \vee_{j \notin I} cond_j)$ *is valid.*

Request chaining means that it exists a request which in conjunction with some rule conclusions triggers few other rules. A system R does not have the chaining property if and only if there is no request chaining.

Definition 4.3 (Types of policy system)**.** *R a policy system is of*

Type 1: *if it is a rule system, without the chaining property, $n \geq 2$ and $conc_i$ is from a finite discrete language containing for instance, $\{\mathbb{PERMIT}, \mathbb{DENY}\}$, some roles, locations, or actions.*

Type 2: *if it is a rule system, $n \geq 2$ and $conc_i$ is an expression containing atoms for authorization, role, obligations, etc. R does not satisfy the chaining property.*

Type 3: *if it is a rule system and $n \geq 1$ rules.*

Type 4: *if it is a set of logical sentences not only implication rules.*

Type 3 and 4 have the same expressive power, which is strictly greater than Type 2 and which is in turn greater than Type 1.

4.3 Level Analysis

We formalize a first notion of conflict in rule systems as follows.

Definition 4.4 (Conflict)**.** *Let R a rule system, with $n \geq 2$ rules, a conflict is an undefined request req such that $\exists I$, $(req => \wedge_{i \in I} cond_i)$ is valid and $\wedge_{i \in I} conc_i$ is unsatisfiable.*

	Type 1	Type 2	Type 3	Type 4
Level 1	√			
Level 2	√	√	√	
Level 3	√	√	√	
Level 4	√	√	√	√
Level 5	√	√	√	√

Table 2: Type and Level constraints

This definition overlaps the formal definitions of conflict we can found in [55, 70, 67, 60, 72, 5, 54, 32, 39, 21] and inconsistency in [49, 62]. We provide in the definition below a formalization of the five principles of Section 3 as Levels 1 to 5.

Definition 4.5 (Levels formalization)**.**

Level 1: *$\exists i, j, \in \{1 \,..\, n\}$, $i \neq j$, $r_i \in R$, $r_j \in R$ and $(cond_i \ \wedge \ cond_j)$ is satisfiable, $conc_i$ and $conc_j$ are incompatible.*

Level 2: *there is a conflict between two rules.*

Level 3: *It exists a conflict between some rules.*

Level 4: *$\exists req, rep_1, rep_2$, req satisfiable, $((R \wedge req) => (rep_1 \ \wedge \ rep_2))$ is valid and $(rep_1 \ \wedge \ rep_2)$ is unsatisfiable.*

Level 5: *$\forall \phi$ a sentence, $R => \phi$ and $R => \neg\phi$ are valid.*

Level 1 is a specific case of conflict. A conflict is an undefined request in a rule system which can be detected by levels lesser or equal to 3. Types of policy system should be analyzed with Levels as in Table 2. The important thing is that Levels 1 to 3 only apply to rule systems while Levels 4 and 5 apply to any system.

Definition 4.6 (Total Inconsistency)**.** *R is totally inconsistent if and only if for all req satisfiable, $(R \ \wedge \ req)$ is unsatisfiable.*

Level 5 is the classic notion of logical (total) inconsistency.

Definition 4.7 (Partial Inconsistency)**.** *R is partially inconsistent if and only if it exists an undefined request.*

This is called inconsistency in [1, 2], and incompatibility in [47] which also defines a stricter notion of conflict living between partial and total inconsistency. Satisfiability is a natural requirement and if R is satisfiable then partial inconsistency is equivalent to be not a valid system. Note also that all these levels are decidable properties as soon as satisfiability is decidable. The previous Levels can be compared as follows.

Lemma 4.8 (Levels Properties)**.** *For any policy system where the Levels apply we have*

- *Level 1 => Level 2 => Level 3 => Level 4.*
- *Level 4 is partial inconsistency.*
- *Level 5 => Level 4.*
- *We do not have: Level 4 => Level 3 => Level 2 => Level 1 and Level 4 => Level 5.*

Proof. The inclusion of the Level 1 in Level 2 is obvious since the constraint on conclusions is enlarged. From Level 2 to Level 3, of course $\{i, j\}$ is a subset of $\{1 \,..\, n\}$. From Level 3 to Level 4: we can take $req = \wedge_{i \in I}\ cond_i$, it is satisfiable. Either $R \wedge req$ is unsatisfiable and then we can derive "anything" from it. Or, if $R \wedge req$ is satisfiable, we can prove $R \wedge req => conc_i$ is valid for all $i \in I$, since R should be rule based, thus grouping by conjunction $conc_i$ in two unsatisfiable replies we get the Level 4. Level 4 is indeed equivalent to $\exists req, \phi,\ req$ satisfiable, $(R \wedge req => \phi)$ and $(R \wedge req => \neg\phi)$ are valid. It says that $R \wedge req$ is unsatisfiable thus R is partially inconsistent.

To justify the last negative property. One difference between Levels 4 and 3 is due to the fact that Level 4 applies to general systems while Level 3 requires rule systems. Even considering a rule system these Levels are still different. It is possible to built a system which is consistent with an undefined request but without conflict for the Level 3. From Level 3 to Level 2, it is possible to built a conflict with n rules but which does not exist with $n-1$ rules. Conflicts of Level 2, due to the unsatisfiability of conclusions are strictly more general than of Level 1. □

Figure 1 graphically represents all these properties in a single picture.

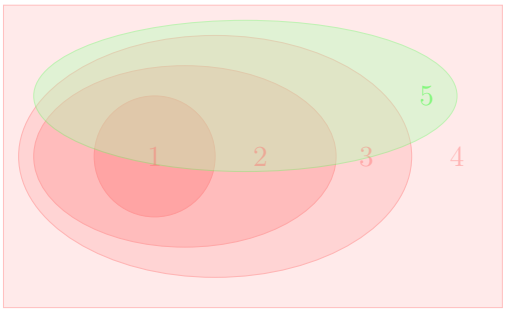

Figure 1: Venn diagram of the Levels

Let us consider the example in Listing 4. It is unsatisfiable (by Levels 5) and of Type 2 but the Level 2 detection will not find any conflict. If we consider the first three rules only, we get a satisfiable system but Level 3 will find a conflict with them. Now if we consider the first two rules it is still a satisfiable system, without conflict but it is possible to built undefined requests caught by the Level 4.

Listing 4: An Unsatisfiable Example

```
(ALWAYS (login(x) AND password(x))) => (SOMETIME enter(x))
(ALWAYS (login(x) AND password(x))) => (SOMETIME read(x))
(ALWAYS (login(x) AND password(x)))
    => ((ALWAYS NOT enter(x))  OR (ALWAYS NOT read(x)))
((SOMETIME NOT login(x)) OR (SOMETIME NOT password(x)))
    => (SOMETIME enter(x))
((SOMETIME NOT login(x)) OR (SOMETIME NOT password(x)))
    => (SOMETIME read(x))
(SOMETIME NOT password(x))
    => NOT ((SOMETIME enter(x)) AND (SOMETIME read(x)))
(SOMETIME NOT login(x))
    => NOT ((SOMETIME enter(x)) AND (SOMETIME read(x)))
```

4.4 Rule systems

Considering rule systems we can be more precise on the classification of real conflicts. The notion of conflict is associated with at least two rules, but we can define a problem of undefined request involving one rule alone. As already noted, to be useful, a rule is not valid thus it raises this new specific case.

Definition 4.9 (1-undefined request)**.** *Let one rule* $(cond => conc)$ *then a 1-undefined request verifies* req *is satisfiable and* $(req => (cond \wedge \neg conc))$ *is valid.*

Lemma 4.10 (Two reduction laws)**.** *Let four propositions* A, B, C, D *these three formulas are equivalent:*

1. $((A => B) \wedge (C => D))$
2. $((A \vee C) \wedge (C \vee \neg B) \wedge (A \vee \neg D)) => (B \wedge D)$
3. $(A \vee C) => ((\neg A \wedge D) \vee (B \wedge \neg C) \vee (B \wedge D))$

The properties 2 and 3 can be used to rewrite a rule system. Using these laws a rule system can be rewritten with more or less rules, but Types 1 and 2 are not stable over these reduction laws. Type 3 is closed under these reduction laws, and using these laws we can equivalently rewrite any rule system to only one rule. In this case, a request is undefined if and only if it is a 1-undefined request. The definition below subsumes the classic notion of conflict in Definition 4.4 and the 1-undefined request.

Definition 4.11 (General conflict)**.** *Let* R *a set of rules then a satisfiable request* req *is a general conflict if and only if* $\exists I, (req => (\wedge_{i \in I} cond_i \ \wedge \neg \wedge_{i \in I} conc_i))$ *is valid.*

A second new kind of undefined request is due to the chaining of two or more rules.

Definition 4.12 (Undefined chaining)**.** *Let* R *a set of rules,* $n \geq 2$*, an undefined chaining is an undefined request with the chaining property.*

We can demonstrate a classification of the undefined requests in a rule system.

Lemma 4.13 (Classification)**.**

1. *If* R *is of Type 1,2 then an undefined request is a general conflict.*
2. *If* R *is of Type 3 then an undefined request is either a general conflict or a chaining but not both.*

Proof. We can remark that req is undefined for a rule system if and only if for all K the expression $req \wedge_{k \in K} \neg cond_k \wedge_{j \notin K} conc_j$ is unsatisfiable. Furthermore, $\vee_{1 \leq i \leq n} \ cond_i$ can be split in disjoint cases and thus it exists a unique maximal K such that $req => \wedge_{k \in K} \ cond_k \wedge_{j \notin K} \ \neg cond_j$ is valid and $req \wedge_{j \notin K} \neg cond_j$ is satisfiable. If we do not have chaining, let req an undefined request $req \wedge_{k \in K} \ conc_k \wedge_{j \notin K} \ \neg cond_j$ is unsatisfiable, hence $(req \wedge_{k \in K} \ conc_k) => \vee_{j \notin K} \ cond_j$ is valid, and $req \wedge_{j \notin K} \ \neg cond_j$ is satisfiable imply that the first condition of chaining $req \wedge_{k \in K} \ conc_k$

is unsatisfiable. From that, we have $req => \wedge_{k\in K}\ cond_k$, and req is a general conflict.

In case of Type 3 system, if $n = 1$ then we have only 1-undefined requests. It is rather obvious that general conflict and undefined chaining are disjoint cases of undefined requests. Considering an undefined request it exists a unique I such that $req => \wedge_{i\in I}\ cond_i\ \wedge_{j\notin I} \neg cond_j$ and $req \wedge_{j\notin I} \neg cond_j$ is satisfiable. If req is satisfiable and not a general conflict we have $req \wedge_{i\in I}\ conc_i$ satisfiable and $((req \wedge_{i\in I}\ conc_i) \wedge_{j\notin I} \neg cond_j)$ is unsatisfiable thus req is a chaining request. □

A notion of (general) n-conflict can be defined as a conflict between exactly n rules, it happens if we have n incompatible conclusions. Thus, in case of simple access control rules with two mutually exclusive literals, if there is a n-conflict with $n > 2$ then there is a 2-conflict. Among the set of requests to submit to a rule system there are some which are not interesting because even if they are defined they will not give a useful result (that is inferring something included in the set of replies). Thus, it is natural to consider requests, written on the language of conditions, and included in $\vee_{1\leq i\leq n} cond_i$ as it is the minimal expression which contains the relevant requests to each rule. We should also remark that if req is undefined it is also included in $\vee_{1\leq i\leq n} cond_i$. In many rule systems the language of the conditions is disjoint from the language of conclusions and their interpretations are separated. This is the most natural assumption to ensure no rule chaining.

Lemma 4.14 (Rule System Hypotheses)**.**

In a rule system, considering requests in $\vee_{1\leq i\leq n} cond_i$ and if the language for conditions is disjoint from the language of conclusions and each conclusion is satisfiable then undefined requests are conflicts in the sense of Definition 4.4.

Table 3: Static detection: Types and Levels.

Reference	Type	Level	Condition	Conclusion
[1]	1	1	PROP	Permission
[2]	1	1	PROP	Permission
[18]	1	1	PROP	Discrete
[66]	1	1	FOL	Permission
[57]	1	1	PROP	Discrete
[60]	1	1	PROP	Permission
[5]	1	1	FOL	Discrete
[54]	1	1	FOL	Discrete
[74]	1	1	PROP	Discrete
[32]	1	1	PROP	Permission
[73]	1	1	PROP	Permission
[39]	1	1	FOL	Permission
[65]	1	1	FOL	Permission
[62]	1	1	PROP	Permission
[21]	1	1	PROP	Permission
[35]	1	5	FOL	Permission
[70]	2	2	FOL	Authorization
[67]	2	2	FOL	Predicate
[20]	2	2	FOL	Normative
[3]	2	2	FOL	Predicate
[43]	2	3	FOL	Predicate
[55]	2	3	FOL	Predicate
[72]	2	4	FOL	Predicate
[49]	3	4	FOL	Predicate
[27]	3	5	FOL	Authorization
[14]	3	5	FOL	Regulatory
[9]	3	5	FOL	Class-Specific
[10]	3	5	FOL	Authorization
[53]	4	5	FOTL	$\asymp$
[41]	4	5	FOL	$\asymp$
[8]	4	5	FOL	$\asymp$
[61]	4	5	FOTL	$\asymp$

4.5 Types and Levels Instances

We present in Table 3 instantiations coming from our previous review. The condition and conclusion columns are related to vocabulary for writing policy conditions and conclusions. *Permission* is $\{\mathbb{PERMIT}, \mathbb{DENY}\}$ and *Discrete* is a set of literals, for instance defining a set of action permissions or roles with hierarchy. *Authorization* are predicates for permit and deny, *Normative* adds the notion of obligation, *Class-Specific* adds data protection while *Regulatory* enriches it with many more predicates like doing action, fulfilled, ceased, or violation. *Predicate* denotes a set of specific predicates. The language disjunction is often clear, but not always made explicit and should be inferred from the context. Examples

classified in Type 3 explicitly discuss the possibility of chaining, except [49]. The embedding of a referenced model is sometime straightforward with few simple transformations, [1, 27, 35, 14, 70, 57, 5, 9, 74, 20, 39, 65, 61, 62, 21]. In some other cases we rely on the examples described in the article [41, 66, 74, 32, 73]. [2] defines a type system for a rule system obtained by extrapolation of the original system. [54] does not use $\mathbb{FOL}$ but precisely it considers SWRL rules with roles and actions as conclusions. [55, 43, 67] are based on similar models where conflicts are related to conditions, obligations, or purposes. The relational model in [72] can be viewed as rules with conditions on roles, resources, actions and conclusions with predicates. [10] uses Left Fusion logic which is derived from Propositional Interval Temporal Logic and subsumes $\mathbb{LTL}$ with a chop (sequential composition) operator. Such a logic can be embedded in $\mathbb{FOL}$. The $\Delta DSTL(x)$ logic of [53] is easily embedded in $\mathbb{FOTL}$. Regarding [3] the language is based on OWL. [49] uses a rule system without details. [8] precisely relies on description logic and subsumption.

Our formalization makes explicit several formal definitions behind the conflict notion and gives a rigorous way to compare the different types and levels we observed. Looking at the Table 3 we can clarify few issues of our related work analysis. Few papers discuss the fact that searching for 2-conflicts (Levels 1 and 2) only is not sufficient [55, 43, 32, 61]. Indeed it is sufficient if we have conclusions which are pairwise disjoint. This is obviously the case when conclusions are $\mathbb{PERMIT}$ or $\mathbb{DENY}$, but conclusions exclusivity should be formally stated in [18, 57, 74]. [5] does not comment this case, but it allows role hierarchy and thus it is possible to built an example with several incompatible roles not pairwise disjoint. [54] does not consider negative roles, but there is a role hierarchy and the same problem may happen.

With Type 2 examples, Lemma 4.14 generally applies but with some exceptions. Often the satisfiability of each conclusion is ensured by individual predicates or it is explicitly stated as in [55, 43]. Type 2 allows to define inconsistent but conflict free rule systems which is not a desirable case, an example is described in Listing 4. This problem does not occur in [55, 43] since it satisfies all the hypotheses of Lemma 4.14, unsatisfiability implies there is an undefined request which is a conflict caught by the Level 3. [72] rules out this case since it relies on Level 4 and implicitly considers a finite set of resources thus ensuring the testing coverage. [20] has only pairwise disjoint atoms as conclusions thus if the system is unsatisfiable there are only 2-conflicts which are caught by the Level 2. However, [70] does not ensure that the relationships in the system environment and its constraints are consistent. [3] lacks details about the conditions to define policies. It is possible to build a system without 2-conflict but with a conflict between more than 2 rules. There is also no guarantee about the satisfiability of the conclusions. [67] is strongly inspired by previous work of Ni et al. but information are lacking about the conclusions and their satisfiability. The algorithm cannot catch conflicting problems between 3 rules which are not pairwise conflicting.

In case of Type 3, we could have other kinds of undefined request but the reported references are relying on either the Level 4 or the Level 5. In [49] Level 4 catches the undefined requests, however the testing coverage is not proved in this work.

Useful rule systems are neither valid nor unsatisfiable, meaning that they do not trivially process requests into replies. The Level 5 appears

as a natural requirement of any logical system and Level 4 is the most general for "conflict" detection and it applies to any rule system. To control the set of requests to submit to a policy system is a major issue, syntactic constraints are either too constraining or hard to define when we have chaining. One benefit of using rules without chaining is that implementation can be simpler and efficient: searching the rule matching the given request then building the conclusion. Furthermore, compiling such a rule system into a set of exclusive rules can be automated as well as a predicate detecting all the conflicts. But an open question is to achieve similar benefits in case of more complex systems with chaining of rules. Level 4 raises issues related to its efficient implementation, the generation and the localization of the undefined requests. A further analysis of undefined requests in rule-based systems is possible, and for instance [12] shows how to compute all these request while making it possible for some middle-size cases studies. Another example is [13] which studies automatic removing of undefined requests and how to both minimize the size of modifications and optimize the processing time. These remarks pave the way for future work.

5 Conclusion

In their survey [16], the authors argue that: "It is thus rather difficult to determine all possible conflicting conditions in advance and so it is still necessary to detect conflicts at run-time". Indeed, before 2008 there was few work promoting static conflict detection, see [16, 22, 59]. If we look at recent related academic papers: height are proposing a dynamic detection while more than thirty argue for a static one. However, 28% of the static approaches are relying on logical consistency, most of the others expect to check conflict only by comparing pairs of rules (47%). There are also few pragmatic approaches implementing a static approach complemented with a dynamic checking. Reviewing the state of the art in conflict detection, we classify these approaches and we summarize the main critics around the dynamic and testing detection. We specifically analyze and formally compare the static detection principles as they are providing stronger assurance on the policy system. Our analysis clarifies the differences and the notions behind the term conflict relating it with the classic notions of satisfiability and consistency. We provide a classification of policy systems and also a formalization of five detection principles. A part from the common notion of conflict there are also general conflict and undefined chaining. Checking for two conflicting rules is not sufficient in case of more complex conclusions than permit and deny. Checking several rules for conflicts is inefficient and a too syntactic approach, semantic approaches are better since stable over the system descriptions. Our comparison gives precise conditions to apply the conflict detection principles and opens few new research perspectives.

References

[1] R. Abassi and S. G. E. Fatmi. An automated validation method for security policies: The firewall case. In M. Rak, A. Abraham, and V. Casola, editors, *Proccedings of the Fourth International Conference on Information Assurance and Security*, pages 291–294. IEEE Computer Society, 2008.

[2] K. Adi, Y. Bouzida, I. Hattak, L. Logrippo, and S. Mankovski. Typing for con-

flict detection in access control policies. In G. Babin, P. G. Kropf, and M. Weiss, editors, *E-Technologies: Innovation in an Open World*, volume 26 of *Lecture Notes in Business Information Processing*, pages 212–226. Springer, 2009.

[3] M. S. Aphale, T. J. Norman, and M. Şensoy. Goal-directed policy conflict detection and prioritisation. In H. Aldewereld and J. S. Sichman, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems VIII*, pages 87–104. Springer Berlin Heidelberg, 2013.

[4] M. Aqib and R. A. Shaikh. Analysis and comparison of access control policies validation mechanisms. *I.J. Computer Network and Information Security*, 7(1):54–69, 2015.

[5] A. Armando and S. Ranise. Automated and efficient analysis of role-based access control with attributes. In *Data and Applications Security and Privacy XXVI*, pages 25–40, 2012.

[6] A. Barth, J. C. Mitchell, and J. Rosenstein. Conflict and combination in privacy policy languages. In V. Atluri, P. F. Syverson, and S. D. C. di Vimercati, editors, *ACM Workshop on Privacy in the Electronic Society*, pages 45–46. ACM, 2004.

[7] M. Y. Becker, A. Malkis, and L. Bussard. A practical generic privacy language. In S. Jha and A. Mathuria, editors, *ICISS 2010*, volume 6503, pages 125–139. Springer, 2010.

[8] T. D. Breaux, H. Hibshi, and A. Rao. Eddy, a formal language for specifying and analyzing data flow specifications for conflicting privacy requirements. *Requir. Eng*, 19(3):281–307, 2014.

[9] M. M. Casalino, H. Plate, and S. Trabelsi. Transversal policy conflict detection. In *Engineering Secure Software and Systems*, pages 30–37, 2012.

[10] A. Cau, H. Janicke, and B. C. Moszkowski. Verification and enforcement of access control policies. *Formal Methods in System Design*, 43(3):450–492, 2013.

[11] R. Chadha and L. Kant. *Policy-Driven Mobile Ad hoc Network Management*, chapter Policy Conflict Detection and Resolution, pages 99–131. Wiley-IEEE Press, 2008.

[12] Z. Cheng, J.-C. Royer, and M. Tisi. Efficiently characterizing the undefined requests of a rule-based system. In C. A. Furia and K. Winter, editors, *Integrated Formal Methods Proceedings*, volume 11023 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2018.

[13] Z. Cheng, J.-C. Royer, and M. Tisi. Removing problems in rule-based policies. In G. Dhillon, F. Karlsson, K. Hedström, and A. Zúquete, editors, *ICT Systems Security and Privacy Protection*, pages 120–133. Springer International Publishing, 2019.

[14] R. Craven, J. Lobo, J. Ma, A. Russo, E. C. Lupu, and A. K. Bandara. Expressive policy analysis with enhanced system dynamicity. In W. Li, W. Susilo, U. K. Tupakula, R. Safavi-Naini, and V. Varadharajan, editors, *Proceedings of the 2009 ACM Symposium on Information, Computer and Communications Security*, pages 239–250. ACM, 2009.

[15] F. Cuppens, N. Cuppens-Boulahia, J. García-Alfaro, T. Moataz, and X. Rimasson. Handling stateful firewall anomalies. In D. Gritzalis, S. Furnell, and M. Theoharidou, editors, *Information Security and Privacy Research*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 174–186. Springer, 2012.

[16] N. C. Damianou, A. K. Bandara, M. S. Sloman, and E. C. Lupu. A survey of policy specification approaches. Technical report, Imperial College of Science Technology and Medicine, London, 2002.

[17] A. Datta, J. Blocki, N. Christin, H. DeYoung, D. Garg, L. Jia, D. K. Kaynar, and A. Sinha. Understanding and protecting privacy: Formal semantics and principled audit mechanisms. In S. Jajodia and C. Mazumdar, editors, *Information Systems Security*, volume 7093 of *LNCS*, pages 1–27. Springer, 2011.

[18] S. Davy, B. Jennings, and J. Strassner. The policy continuum-policy authoring and conflict analysis. *Computer Communications*, 31(13):2981–2995, 2008.

[19] A. Degtyarev, M. Fisher, and B. Konev. Monodic temporal resolution. *ACM Transactions on Computational Logic*, 7(1):108–150, Jan. 2006.

[20] R. Delmas and T. Polacsek. Formal methods for exchange policy specification. In C. Salinesi, M. C. Norrie, and O. Pastor, editors, *CAiSE*, volume 7908 of *LNCS*, pages 288–303. Springer, 2013.

[21] F. Deng and L.-Y. Zhang. Elimination of policy conflict to improve the PDP evaluation performance. *J. Network and Computer Applications*, 80:45–57, 2017.

[22] N. Dunlop, J. Indulska, and K. Raymond. Methods for conflict resolution in policy-based management systems. In *Enterprise Distributed Object Computing Conference*, pages 98–111. IEEE Computer Society, 2003.

[23] K. Fatema and D. W. Chadwick. Resolving policy conflicts - integrating policies from multiple authors. In L. S. Iliadis, M. P. Papazoglou, and K. Pohl, editors, *Advanced Information Systems Engineering Workshops - CAiSE International Workshops*, volume 178 of *Lecture Notes in Business Information Processing*, pages 310–321. Springer, 2014.

[24] M. Fisher. *An Introduction to Practical Formal Methods using Temporal Logic*. Wiley, 2011.

[25] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *International Conference on Software Engineering*, 2005.

[26] J. García-Alfaro, N. Boulahia-Cuppens, and F. Cuppens. Complete analysis of configuration rules to guarantee reliable network security policies. *Int. J. Inf. Sec*, 7(2):103–122, 2008.

[27] J. Y. Halpern and V. Weissman. Using first-order logic to reason about policies. *ACM Transactions on Information and System Security*, 11(4):1–41, July 2008.

[28] W. Han and C. Lei. A survey on policy languages in network and security management. *Computer Networks*, 56(1):477–489, Jan. 2012.

[29] A. Hanamsagar, N. Jane, B. Borate, A. Wasvand, and S. Darade. Firewall anomaly management: A survey. *International Journal of Computer Applications*, 105(18):1–5, 2014.

[30] H. Hu, G.-J. Ahn, and J. Jorgensen. Detecting and resolving privacy conflicts for collaborative data sharing in online social networks. In R. H. Zakon, J. P. McDermott, and M. E. Locasto, editors, *Computer Security Applications Conference*, pages 103–112. ACM, 2011.

[31] H. Hu, G.-J. Ahn, and K. Kulkarni. Detecting and resolving firewall policy anomalies. *IEEE Trans. Dependable Sec. Comput*, 9(3):318–331, 2012.

[32] H. Hu, G.-J. Ahn, and K. Kulkarni. Discovery and resolution of anomalies in web access control policies. *IEEE Trans. Dependable Sec. Comput*, 10(6):341–354, 2013.

[33] N. Hu, P. Zhu, H. Cao, and K. Chen. Routing policy conflict detection without violating ISP's privacy. In *CSE*, pages 337–342. IEEE Computer Society, 2009.

[34] C. Huang, J. Sun, X. Wang, and Y. Si. Inconsistency Management of Role Based Access Control Policy. In *Int. Conf. on E-Business and Information System Security*, 2009.

[35] F. Huang, Z. Huang, and L. Liu. A DL-based method for access control policy conflict detecting. In F. Yang, H. Mei, and J. Lv, editors, *Proceedings of the First Asia-Pacific Symposium on Internetware*, pages 16–21. ACM, 2009.

[36] G. Hughes and T. Bultan. Automated verification of access control policies using a SAT solver. *International Journal on Software Tools for Technology Transfer*, 10(6):503–520, Dec. 2008.

[37] J. Hwang, T. Xie, and V. C. Hu. Detection of multiple-duty-related security leakage in access control policies. In *Secure Software Integration and Reliability Improvement*, pages 65–74. IEEE Computer Society, 2009.

[38] H. Janicke, A. Cau, F. Siewe, and H. Zedan. Dynamic access control policies: Specification and verification. *The Computer Journal*, 56(4):440–463, Apr. 2013.

[39] H. Jebbaoui, A. Mourad, H. Otrok, and R. A. Haraty. Semantics-based approach for detecting flaws, conflicts and redundancies in xacml policies. *Computers & Electrical Engineering*, 44:91–103, 2015.

[40] V. R. Karimi, P. S. C. Alencar, and D. D. Cowan. A formal modeling and analysis approach for access control rules, policies, and their combinations. *Int. J. Inf. Sec*, 16(1):43–74, 2017.

[41] V. R. Karimi and D. D. Cowan. Verification of access control policies for REA business processes. In S. I. Ahamed, E. Bertino, C. K. Chang, V. Getov, L. Liu, H. Ming, and R. Subramanyan, editors, *COMPSAC, Volume 2*, pages 422–427. IEEE Computer Society, 2009. 978-0-7695-3726-9.

[42] N. Khairdoost and N. Ghahraman. Term rewriting for describing constrained policy graph and conflict detection. In *Progress in Informatics and Computing (PIC)*, pages 645–651. IEEE, 2010.

[43] Y. Kim and E. Song. Privacy-aware role based access control model: Revisited for multi-policy conflict detection. In *Information Science and Applications*, pages 1–7. IEEE, 2010.

[44] R. Kuhlisch. A description model for policy conflicts for managing access to health information. In B. Lantow, K. Sandkuhl, and U. Seigerroth, editors, *International Workshop on Information Logistics, Knowledge Supply and Ontologies in Information Systems*, volume 1028, pages 44–55. CEUR-WS.org, 2013.

[45] R. Kuhlisch. Modeling and recognizing policy conflicts with resource access requests on protected health information. *Complex Systems Informatics and Modeling Quarterly*, 11:1–19, 2017.

[46] N. Li, Q. Wang, W. H. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin. Access control policy combining: theory meets practice. In B. Carminati and J. Joshi, editors, *ACM Symposium on Access Control Models and Technologies*, pages 135–144. ACM, 2009.

[47] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. Exam: a comprehensive environment for the analysis of access control policies. *Int. J. Inf. Sec*, 9(4):253–273, 2010.

[48] M. Ludwig and U. Hustadt. Implementing a fair monodic temporal logic prover. *AI Commun*, 23(2-3):69–96, 2010.

[49] J. Ma, D. Zhang, G. Xu, and Y. Yang. Model checking based security policy verification and validation. In *2nd Int. Workshop on Intelligent Systems & Applications*, pages 1–4. IEEE Computer Society, 2010.

[50] A. Margheri, M. Masi, R. Pugliese, and F. Tiezzi. Developing and enforcing policies for access control, resource usage, and adaptation - A practical approach -. In *Web Services and Formal Methods*, pages 85–105, 2013.

[51] A. Margheri, R. Pugliese, and F. Tiezzi. On properties of policy-based specifications. In M. H. ter Beek and A. Lluch-Lafuente, editors, *International Workshop on Automated Specification and Verification of Web Systems*, volume 188 of *EPTCS*, pages 33–50, 2015.

[52] A. Mohan, D. M. Blough, T. M. Kurç, A. R. Post, and J. H. Saltz. Detection of conflicts and inconsistencies in taxonomy-based authorization policies. In F.-X. Wu, M. J. Zaki, S. Morishita, Y. Pan, S. Wong, A. Christianson, and X. Hu, editors, *IEEE International Conference on Bioinformatics and Biomedicine*, pages 590–594. IEEE Computer Society, 2011.

[53] C. Montangero, S. Reiff-Marganiec, and L. Semini. Logic-based conflict detection for distributed policies. *Fundamantae Informatica*, 89(4):511–538, 2008.

[54] M. A. Neri, M. Guarnieri, E. Magri, S. Mutti, and S. Paraboschi. Conflict detec-

tion in security policies using semantic web technology. In *Satellite Telecommunications (ESTEL)*, pages 1–6. IEEE, 2012.

[55] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C.-M. Karat, J. Karat, and A. Trombeta. Privacy-aware role-based access control. *ACM Trans. Inf. Syst. Secur.*, 13(3):24:1–24:31, July 2010.

[56] Y. Ni. Policy conflict detection and resolution in goal policy-driven management. *IJWMC*, 7(5):495–499, 2014.

[57] S. Niksefat and M. Sabaei. Efficient algorithms for dynamic detection and resolution of IPSec/VPN security policy conflicts. In *AINA*, pages 737–744. IEEE Computer Society, 2010.

[58] OASIS Standard. eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`, 2013.

[59] T. Phan, J. Han, J.-G. Schneider, T. Ebringer, and T. Rogers. A survey of policy-based management approaches for service oriented systems. In *Australian Software Engineering Conference*, pages 392–401. IEEE Computer Society, 2008.

[60] Y. Ren, F. Cheng, Z. Peng, X. Huang, and W. Song. A privacy policy conflict detection method for multi-owner privacy data protection. *Electronic Commerce Research*, 11(1):103–121, 2011.

[61] J.-C. Royer and A. Santana De Oliveira. AAL and static conflict detection in policy. In *15th International Conference on Cryptology and Network Security*, LNCS, Milan, Italia, Nov. 2016. Springer.

[62] R. A. Shaikh, K. Adi, and L. Logrippo. A data classification method for inconsistency and incompleteness detection in access control policy sets. *Int. J. Inf. Sec*, 16(1):91–113, 2017.

[63] R. A. Shaikh, K. Adi, L. Logrippo, and S. Mankovski. Inconsistency detection method for access control policies. In *Information Assurance and Security*, pages 204–209. IEEE, 2010.

[64] L. Shen, Z. Wang, X. Zhang, and J. Gu. Study on the policy conflict detection in the security management model. In *Signal Processing, Communications and Computing*, pages 1–5. IEEE, 2015.

[65] M. St-Martin and A. P. Felty. A verified algorithm for detecting conflicts in XACML access control rules. In J. Avigad and A. Chlipala, editors, *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, Saint Petersburg, FL, USA, January 20-22, 2016*, pages 166–175. ACM, 2016.

[66] B. Stepien, S. Matwin, and A. P. Felty. Strategies for reducing risks of inconsistencies in access control policies. In *Availability, Reliability, and Security*, pages 140–147. IEEE Computer Society, 2010.

[67] L. Sun, H. Wang, X. Tao, Y. Zhang, and J. Yang. Privacy preserving access control policy and algorithms for conflicting problems. In *TrustCom*, pages 250–257. IEEE Computer Society, 2011.

[68] S. Thanasegaran, Y. Tateiwa, Y. Katayama, and N. Takahashi. Simultaneous analysis of time and space for conflict detection in time-based firewall policies. In *CIT*, pages 1015–1021. IEEE Computer Society, 2010.

[69] F. Turkmen, J. den Hartog, S. Ranise, and N. Zannone. Analysis of XACML policies with SMT. In R. Focardi and A. C. Myers, editors, *Principles of Security and Trust*, volume 9036 of *LNCS*, pages 115–134. Springer, 2015.

[70] Y. Wang, H. Zhang, X. Dai, and J. Liu. Conflicts analysis and resolution for access control policies. In *Proceedings of Int. Conf. on Information Theory and Information Security (ICITIS)*, pages 264–267. IEEE Computer Society, 2010.

[71] B. Wu, X. yuan Chen, Y. fui Zhang, and X. dong Dai. An extensible intra access control policy conflict detection algorithm. In *Computational Intelligence and Security*, pages 483–488. IEEE Computer Society, 2009.

[72] X. Xia. A conflict detection approach for XACML policies on hierarchical resources. In *Conference on Green Computing and Communications, Conference on Internet of Things, and Conference on Cyber, Physical and Social Computing*, pages 755–760. IEEE Computer Society, 2012.

[73] D. Yan, J. Huang, Y. Tian, Y. Zhao, and F. Yang. Policy conflict detection in composite web services with RBAC. In *International Conference on Web Services*, pages 534–541, 2014.

[74] C. Yuan, X. Liang, Y. Bo, and C. Xia. Research on computer network defense policy conflict detection. In *Information and Communication Technologies (WICT)*, pages 1193 – 1197. IEEE, 2012.

A Analyzed References

A.1 Surveys

We provide a short review of the survey from [16] mainly for historical reasons. It helps in understanding the evolution in this domain. This paper reviews a set of policy languages and some of their characteristics. A short informal discussion is made on the notion of conflict and the way to solve them. Existing conflict resolution means were: errors, priority, and meta-policies. There is no precise description of algorithms for conflict detection and clearly this feature was missing in most of the proposed tools. At this time, the authors argue that runtime detection is required and that not all conflicting situations can be known in advance.

[22] is a general survey of static and dynamic conflict detection and resolution in large open distributed systems. They also consider that not all conflict can be statically detected and resolved. A set of combining algorithms for dynamic resolution is not sufficient at all.

The book chapter [11] provides an overview of the state of the art in policy conflict detection and resolution. The authors propose a classification of policy rules in Event-Condition-Action, access control and configuration policies and argue that different stages of conflict detection are needed. One important feature is the fact that policies are modified or not at runtime. Thus runtime detection of application specific policy conflict is required, except for the most trivial

cases like modality conflicts. They claims that they are obvious reasons to do conflict detection at runtime. However, they do not precisely describe the existing algorithms.

In their survey, [59], the authors study five policy frameworks in order to evaluate their suitability for the management of service oriented systems. Namely the frameworks are Ponder, IETF, KAoS, Rei and WS-Policy. However, in their review none detection algorithm is mentioned.

[28] is a general survey about policy management in network systems. This paper investigates mainly existing work, language proposals and discusses the key issues. Among these, the first issue is policy conflict detection and resolution. Still it emphasizes the fact that some conflicts can be detected statically by translating policies in a logical framework and some other only dynamically and resolved by meta-policies.

[4] is the most recent and exhaustive survey of access control policy systems and their validation mechanisms. It classifies the various approaches regarding the validation framework used: formal methods, mining techniques and so on. It proposes a general table of the published work with their main characteristics: like inconsistency, incompleteness, static or dynamic and so on. However, it is informal and the conflict definition is weak and does not exhibit that, for instance, inconsistency checking is varying from one approach to another. We need a more formal analysis to compare and to show the weaknesses of the various approaches of conflict detection. The authors reviewed 26 papers from 2005 to 2016 and only 23 with a conflict management feature. They do not describe and compare the existing conflict detection principles.

A.2 Dynamic Detection

Some approaches suggest a dynamic detection and generally an automatic resolution. The principle is to analyze, at runtime, each request and to look for conflicting results. In case of conflict a combining algorithm, a meta-policy, or a priority are used to select one reply to the request.

XACML is a de facto standard language for attribute-based access control policies [58]. XACML allows the definition of policies and rules managing authorizations and obligations. A rule will produce an access decision which is *permit, deny, or not-applicable*. Rules and policies can overlap and then can produce contradictory results. In order to solve conflicts, the XACML introduces *combining algorithms* for rules and polices such as: permit-overrides, deny-overrides, or first-applicable.

The authors of [52] consider access control policies with hierarchy of concepts and define an algorithm for dynamic conflict detection. They consider a classic notion of conflict but claim it is distinct from an inconsistency which is related to data relationship inference. The exact rule language is not described and it is not also clear which part of XACML is covered by this analysis.

An interesting compositional framework is proposed in [38]. It allows the specification and the verification of access control policies where history and temporal aspects are taken into account. Interval temporal logic is used for defining the semantics of the formal policies and to reason about them. The language, called SANTA, supports positive and negative policies as well as decision rules to resolve conflicts during the evaluation. Thus we consider that the detection is only dynamic.

[23] analyzes the principle of combining algorithm in XACML. The authors show that con-

flicting situation occurs with subject owning different roles. They propose a resolution mechanism with an algorithm choosing the combining rule dynamically.

Few papers [36, 51, 69] propose a formal way to verify XACML by translating it in a more formal language and then use a verification tool support. In principle these approaches could statically check for conflicts but they did not do it since they resolve the real conflicts using combining algorithms which state rule precedence in case of conflicts. Thus they rely on dynamic conflict detection as XACML.

A.3 Testing Detection

In this case the method is to explicitly and statically generate a set of requests and to test if a conflict occurs using the normal request evaluation process.

[37] focuses on multiple-duty and conflicts in access control rules. The authors explain that multiple roles may lead to conflicts but the resolution mechanism choose a solution which is not the expected one because of the rule ordering. The authors proposes a framework to assist the detection of multiple roles that can leads to potential leakages. They generate a finite set of requests and check for their consistency. It relies on Margrave [25] which is realizing the detection task but this tool only analyzes a subset of XACML.

[32] focuses on anomalies detection in web policies using XACML. These anomalies are potentially numerous due to the numbers of rules, administrators, resources, actions, and subjects. The authors consider that statically removing the conflicts is difficult. The authors think that a global analysis is needed in order to detect all conflicts and redundancies between more than two rules. The technique uses two levels of partition (rules and requests) and consider all the request space of a rule. They do not consider obligations and discrete time.

A.4 Static Detection

In this case the principle is to rely on an algorithm, at design time, to statically check for conflicts and without an explicit generation and evaluation of the requests.

[1] is an example of work which explicitly applies software engineering techniques to security policies. They consider a formal model inspired from RBAC (Role Based Access Control) and the Promela language, it allows authorizations, obligations and negative obligations but without parameters, only finite sets are considered. The consistency checking relies on the idea of checking conflicting pairs of rules. However, while the language allows obligations and negative obligations, the checking is restricted to conflicting authorizations only.

[18] advocates for a formal model of policy and an automated tool support. The policy model is based on an object-oriented information model, and several policy levels linked by a refinement relation. The algorithm to check for conflicts is complex because the model and its management are rich. However, the core principle to analyze a candidate policy for potential conflict is done on a pair-wise basis with all the other deployed policies. The language allows finite enumeration of entities only and the algorithm is based on a matrix computation.

In [27], the authors describe a way to reason about policies and their consistency. They define the Lithium language, and can statically detect conflicts. They only consider a flexible pure access control language with permissions

and prohibitions, it supports neither linear temporal time nor obligations. The principle to check conflict relies on the use of consistency in many-sorted $\mathbb{FOL}$, checking that a property and its negation are valid.

[53] mostly focuses on telecommunication systems and service oriented systems where policies are distributed. The paper discusses policy conflict and a detection mechanism. APPEL is a policy language with an informal syntax and based on the Event-Condition-Action paradigm with flat domain of locations but no classic deontic modalities for access control. The language introduces some rule operators expressing conditions, ordering and parallelism. Its formal semantics is expressed in ΔDSTL(x) a first-order logic with localities and inspired from Unity. It allows a future operator (leads-to) and a past one (because).The authors takes a logic-based approach: conflicts are detected by deducing specific formulae in a suitable theory. The conflict definition is domain dependent and defined by the user. However, the method remains paper based and no automatic support is provided.

[2] analyzes some access control models. The authors argue for a static detection of conflicts. In case of conflict it is important to alert the privacy or security officer because none algorithm is able to resolve the conflict. It proposes an access control model with permission and prohibition, first-order conditions, dynamic groups for users, resources, actions and a type system which ensures the absence of conflict. The model does allow neither obligation nor temporal logic. The checking principle is stated as: "two user groups that have common elements should not have different access rights".

[14] argues that a policy language should be expressive, dynamic and supported by a rich set of tools. The authors discuss the need of access control and obligation features but also temporal aspects and complex dependencies between authorizations and obligations. They list a set of required analyses, among them conflict detection and coverage. The Event Calculus enables to describe how events and actions occurring change the system state. This provides a flexible and rather uniform framework. They use abductive, constraint logic programming as the basis of algorithms. A prototype implementation is available but work only with finite domains. Modal conflicts are checked by proving that we do not have both permitted and denied conclusions.

The approach of [41] is to introduce access control policies in a business process. This business process is based on the Resources-Events-Agents (REA) model and encoded in Alloy. Alloy allows to perform various verifications on a formal model, for instance clause inconsistency.

[34] considers the management of RBAC policies enriched with role hierarchies, separation of duty constraints and cardinality constraints. A static detection algorithm is informally described and based on transitive matrices, on the graph of roles, and the use of the Tarjan's algorithm. This algorithm only considers finite sets of entities.

[35] studies XACML access control policies with hierarchies of roles and resources. The propagation rules along the hierarchies do not allow positive authorizations to propagates thus leading to potential conflicts. The authors translate their policies in description logic and claims that conflict detection can be considered as checking consistency of ABox.

A graph model (CPG) is sketched in [42] which allows access control, nested policy and information transfer. The analysis of the model is done by a translation into a term rewriting system. The authors defines the translation of a constrained policy graph as a term rewriting system

which is confluent and terminating. Thus each access request results in a normal form, grant or deny and not both of them. The notion of conflict is defined by two rules with unified conditions and inconsistency in permissions. The notion of consistency for term rewriting system is also introduced for requests which do not have contradictory replies.

The paper [43] studies the principle of pairs of conflicting rules in a privacy aware RBAC model. The model allows only positive assignment and a two-conflict is defined as the intersection of two policy conditions. The authors shows that it is not sufficient to find conflict between two rules and propose several algorithms. The performance analysis shows that the number of rules and the detection with more than two rules entails efficiency.

In [49], the authors define and formalize the validity and reliability concepts. Then, they propose a model checking based method to validate the consistency and completeness for security policies. They rely on the common idea to check for two contradicting rules to detect conflicts.

[57] considers conflict in policies for virtual networks. The proposed approach detects conflict as soon as a new rule is added, it is static since independent from a particular request. The detection method is based on checking two rules for conflict and use the BDD (Binary Decision Diagram) encoding.

[55] presents a framework supporting a privacy-aware access control mechanism. The authors formally define the notion of privacy-aware permissions and the notion of conflicting permission assignments in P-RBAC, together with efficient conflict-checking algorithms. They introduce two kinds of conflicting permission assignment. The first with conflicting conditions is due to implicit negative permission and the matching of a condition with the negation of another condition. The second results from ambiguous rules, that is, the same access request causes a different obligation to be invoked. The authors identify that a simple checking of pairs of rule is not sufficient since a conflict may arise between three rules while there is no conflict between pairs. Thus such a correct conflict checking should consider all the policies in the system and this entails efficiency.

[63] recognizes the importance of static conflict detection and argues for a new method more efficient than logical approaches. The authors proposes to reuse an algorithm from machine learning. The language supports access control and delegation but neither temporal logic nor obligation. The notion of inconsistency is used, "We have a direct inconsistency when two rules present in the same policy set lead to contradictory conclusions.", and it is equated with the notion of conflict.

[66] states the "inadequacy of conflict resolution mechanism" and explains that the complexity of the mechanism comes from historical and legacy reasons. The notion of conflict is "values that satisfy two rules with opposite effects" and the checking is pair-wise based. They demonstrate that the solution is applicable and Prolog can be used to look for conflicts.

[70] identifies three kinds of conflicts: modality conflict, redundancy conflict and potential conflict. The last one corresponds to the usual notion of conflict due to intersection of conditions in rules with opposite effects. The model is access control without obligation and discrete time but with conditions on the subjects and roles hierarchy. Potential conflicts are related to pairs of contradictory rules but the work does not define an algorithm to detect them.

The authors of [30] focus on collaborative data sharing and privacy conflicts in social networks. The conflict notion is informally states as: "when two users disagree on whom the shared data item should be exposed to, we say a privacy conflict occurs". A space segmentation algorithm is used to partition accessor spaces of a shared data item into disjoint segments. Then it uses the segmentation to identify conflicting segments. This is a specific approach to the problem dealing with finite sets and limited features.

In the context of private data shared among several owners, like email, it is important to early detect policy conflicts. The authors of [60] propose a model for the policy set based on a stratified graph. Privacy conflict represents the semantic conflict mismatches between two policies and it is also represented as a graph. Then a graph isomorphism operation is defined in order to check if the policy set has a conflict.

The work from [67] proposes a framework for privacy preserving access control policies, and describes algorithms for conflicting problems taking into account purposes, conditions and obligations. It uses a rich access control model based on usage access control and enriched to control data disclosure. Purposes, allowed or prohibited, are organized in a tree which varies dynamically. Obligations are possible, there is no parameter and no linear time. A detailed analysis of the purpose notion is presented as well as a conflict algorithm based on the analysis of pairs of rules.

[9] studies transversal conflicts occurring between different classes of policy. The authors propose a general approach based on an abstract domain description model, and specific policy predicates expressing authorization, obligation, filtering, firewall, and data protection concerns. All this information is expressed in $\mathbb{FOL}$ and with logical rules. The conflicts are detected by expressing the expected property and then using consistency checking. One strong limitation is that the method is mainly manual without explicit tool support.

The work from [54] proposes a semantic web approach to the detection and resolution of policy incompatibilities. The model focuses on authorization incompatibilities using a formal definition of modality conflict based on pairs of system authorizations. The authors advocates for a semantic web approach since it is better integrated with existing XML and Web technology. Authorization rules are described in SWR and an OWL reasoner is used.

[72] argues for a static conflict detection and identifies two kinds of conflicts: authorizations and resource conditions. The language is a restricted version of RBAC but with hierarchy of resources. The proposed approach is rather limited: no obligation, no hierarchy on roles, and no linear time. This work focuses on conflicts occurring due to policy evolution. The principle is to check for the existence of resources or permissions which are conflicting. The approach uses model-checking in a context with hierarchical resources, but it is used as a technical trick to find paths in the DAG representing this hierarchy.

In [5] the authors argue for a formal analysis and define an RBAC model extended with dynamic roles and negative roles. From that they formally state two crucial problems: detection of redundancies and detection of conflicts. They propose an SMT (Satisfiability Modulo Theory) approach to check the condition satisfiability of pairs of incompatible rules. It assumes that rules are only authorizations with conjunction of atoms.

The computer network defense policy conflict detection model (CNDPDM) is introduced in [74]. A classification of conflicts is pro-

posed with few formal definitions. As it appears all these cases are instances of the well-known schema: overlapping conditions in two policies with incompatible measures. The syntax of conditions as well as the way to judge for their comparisons are lacking. However, examples suggest that simple predicates and simple rules are sufficient enough.

[3] addresses the challenge of providing automated support for identifying and resolving conflicts. The authors note that: "It is difficult, even for experts, to write consistent, unambiguous and accurate policies, and conflicts are practically unavoidable". They distinguish between logical and functional conflicts. The first case is related to conflicts like permitted and denied actions. The second is rather application or domain dependent conflict as proposed by some others. Both cases are defined between a pair of rules and a checking algorithm is proposed.

[10] presents a formalization of access control policies in Fusion logic and an enforcement mechanism based on a verification procedure. In this case the resolution is dynamically done by some specific meta-rules but the detection is statically done. The policies are history-based and this makes verification more challenging. The language allows access controls but without obligation. The authors analyze a suitable candidate for the temporal semantics and choose Left Fusion logic a subset of the propositional interval temporal logic. Verification is bounded and based on translation to BDD. Conflict detection is based on the satisfiability of checking positive and negative authorizations.

In the context of privacy and exchange information [20] is a good example of a static conflict detection approach. The authors define a formal language integrating deontic concepts of obligation, permission and prohibition. They study various properties and the consistency one is related to the detection of conflicts. They consider that consistency is no conflict and they propose a simple algorithm to detect conflicts between pairs of rules. The condition satisfiability is checked thanks to bounded model-checking.

In the context of healthcare [44] proposes a semantic model for privacy policies. The paper introduces privacy, access control and disclosure policy rules but without a precise syntax. This work considers the conflicts between two rules and a classification of the conflicts. A representation of the rules is suggested with RDF but without a real tool support.

In the context of social networks and privacy concerns, the conflict detection has been explored in [8]. The deontic constructions, obligation, permission and interdiction are taken into consideration. The proposed formalism, based on description logic, enables conflict detection between what is permitted and what is prohibited. The authors reduce the conflict compliance of two policies to description logic subsumption and present results from performing automated conflict detection within the Facebook, Zynga, and AOL privacy specifications. Indeed they are interested in conflicts in policy compliance, but the approach can potentially check for conflicts in policies and this point is discussed in the paper.

[73] introduces the model of CWS-RBAC and authorization policy implemented in this model. This is an extension of the RBAC model to cope with web services composition. Algorithms basically compare pairs of policy but taking into account role hierarchy, object role composition, and time constraints.

[65] exposes the critical importance of the static conflict detection and defines an algorithm for that. This work considers a subset

of XACML, access control with real time constraints and look for pairs of conflicting rules. It focuses on the time constraints and their conflicts without other conditions. The main novelty is the proof of the algorithm processed thanks to the Coq prover.

AAL [61] for Abstract Accountability Language allows to write accountability clauses and as such it overlaps many classic access control, privacy and security languages. The authors argue that the static detection of conflicts should be based on logical consistency or satisfiability. They concretely illustrate that the approach is successful, more powerful and general than some existing approaches.

A.5 Mixed Detection

Few proposals consider that conflict detection should be done at several stages, mainly static as possible and dynamic for the remaining conflicts.

[64] claims: "it is very important to do static conflict detection when the policies are formulated and revised, and to do dynamic conflict detection when the system is running". This paper discusses informally the requirements for a good policy language. It proposes a simple policy model with modalities (permission, obligation and forbidden) and tense features (always, immediate, before, etc). The authors suggests to use static conflict detection, that is checking if two policies have overlapping conditions. It also defines a conflict library which dynamically checks for various conflicts (for example tense conflicts).

A policy-driven approach is appealing for system management, it enables administrators to express their management intentions and constraints. [56] uses a two levels approach with goal policies and a refined action level more operational. It considers that dynamic detection for action policies is needed because of environmental conditions but also a static approach to find out and resolve the potential conflicts. The checking principle examines the overlapping of subjects, goals and targets of two goal policies, and then decide whether there is a conflict.