Policy-Based Access Control from Numerical Evidence [1]

Jason Crampton
Information Security Group, Royal Holloway, University of London
Egham, Surrey, TW20 0EX, United Kingdom
jason.crampton@rhul.ac.uk

Michael Huth
Department of Computing, Imperial College London
London, SW7 2AZ, United Kingdom
{m.huth, jimhkuo}@imperial.ac.uk

Charles Morisset
Centre for Cybercrime and Computer Security, Newcastle University
Claremont Tower, Newcastle, NE1 7RU, UK
Charles.Morisset@newcastle.ac.uk

4 October 2013

Abstract

Increasingly, access to resources needs to be regulated or informed by considerations such as risk, cost, and reputation. We therefore propose a framework for policy languages, based on semi-rings, that aggregate quantitative evidence to support decision-making in access control systems. As aggregation operators "addition", "worst case", and "best case" over non-negative reals are both relevant in practice and amenable to analysis, we study an instance, Peal, of our framework in that setting. Peal is a stand-alone policy language but can also be integrated with existing policy languages.

Peal policies can be synthesized into logical formulae that no longer make reference to quantities but capture all policy behavior. Satisfiability checking of such formulae can be used to validate and analyze policies in this new evidence-based approach. We discuss a number of applications, including vacuity, redundancy, change-impact and safety analysis. The synthesis algorithm requires a form of subset enumeration, for which we develop bespoke algorithms and demonstrate experimentally that our algorithms work better than generic state exploration methods. We also sketch how our approach extends from non-negative reals to other semi-rings and even to rings such as the real numbers.

Keywords: access control, policy languages, formal verification, logical synthesis, trust, reputation, risk.

1 Introduction

Many security-related systems maintain "policies" that are responsible for mapping requests to decisions, where "system", "request" and "decision" are interpreted in a broad sense. Typically such policies include rules of the form "if a request satisfies certain conditions, then return decision

[1] Please cite this research note as *Jason Crampton, Michael Huth, and Charles Morisset.* Policy-Based Access Control from Numerical Evidence. *Technical Report 2013/6, Department of Computing, Imperial College London, ISSN 1469-4174, 2013.*

d". A rule set may be evaluated as a list or as a tree, using operators that define how decisions from individual rules are to be combined. Access control systems are a good example of this, where a set of rules of the above form define which users are authorized for which resources. Other examples include, but are by no means limited to:

- firewalls, where incoming "requests" are packets with external source and internal destination addresses, and outgoing "requests" are packets with internal source and external destination addresses;
- credit rating systems, where the requests model clients seeking approval for financial services and the rules determine creditworthiness;
- trust or reputation management systems, where "requests" model the evaluation of an entity's trustworthiness (e.g. with respect to a particular activity).

In many of these systems – notably trust management systems and credit rating systems – numerical scores play an important part in the computation of the decision. Moreover, it is widely recognized that it is becoming difficult to define policies that can anticipate and provide appropriate responses for all possible requests in increasingly open, dynamic, ubiquitous and interconnected computing milieux. This has led to increased interest in risk- and trust-based access control [6, 9, 10, 11], where access control decisions are determined using numerical values.

There has been considerable research into questions of how to combine trust scores in order to compute a single trust decision (see the survey paper by Jøsang *et al.* [17], for example). However, there has been little work on defining a policy language that explicitly incorporates such scores and that allows for a formal analysis of the impact of both the combination and the change of scores. In this paper, we introduce a framework for defining such policy languages, which can be used in a variety of contexts, not least to enhance existing languages for access control policies.

The policy languages of our framework are rule-based, where rules have as meaning a value in some semi-ring. If a rule is applicable to a request, it provides quantitative evidence (for example of user trustworthiness). Our policy languages then use operators to *aggregate* such evidence: the composition of rules into policies, and of policies into policy sets is done using the operators of one or several semi-rings. This compositional design allows us to aggregate evidence to support an access-control decision in an expressive but uniform manner.

For the sake of concreteness and illustrative purposes, we will mainly use the semi-ring operations min, max, and $+$ over the non-negative reals, but our framework does not preclude the use of other operations. In typical use cases, we would use a policy to express a particular aspect of a wider policy – such as those characteristics and their associated scores that have a negative effect on the overall score – and values returned by these policies would themselves be combined, as discussed above.

We believe that the development of such languages is timely, but we also recognize that these language should be amenable to analysis, so that we can verify that policies are a faithful encoding of the (security) requirements, and intuitive, so that policy authors are able to use the language effectively. To this end, we study here a specific instance of our framework – the language Peal – and consider the extent to which we can translate Peal policies into logical formulae (containing the same predicate symbols but no arithmetic values or semi-ring operations) with identical semantics. This *synthesis* of logical formulae from policies allows us to analyze the original policies and *answer* questions of the form: (a) Does the policy always return the same decision? (b) Does the policy never return a particular decision? (c) Does the decision returned by a policy change if we modify

the scores associated with predicates? These questions and their answers allow a policy author to determine whether a policy meets the intended requirements and to perform change-impact analysis for policies that may express aspects of risk, cost, security, etc. in numerical form.

In the next section we introduce our framework for designing policy languages that aggregate evidence over semi-rings. Peal, the specific language that we study in this paper, is described in Section 3. In Section 4, we define the process for synthesizing logical formulae from Peal policies, describe a synthesis algorithm and introduce a novel algorithm to minimize its performance bottleneck. In Section 5, we describe basic analyses of Peal policies and how we can perform them on synthesized formulas. Related and future work are discussed in Section 6, and the paper concludes in Section 7.

2 General framework for aggregation

In this section, we introduce a framework for developing policy languages that process and aggregate values, where those values are typically quantitative in nature and provide "evidence" for or against requests being granted. The evaluation of a policy yields a quantitative value that is used to inform a decision-making process. The design of the framework is influenced by attribute-based languages and policy algebras [7, 8, 12, 23, 26].

Each policy language is parameterized by a non-empty set of decisions $\mathbb{D}$, a set of binary operators on $\mathbb{D}$, a set of predicates Π, and an evaluation mechanism that is used to associate a truth value with predicates in Π. The set of decisions $\mathbb{D}$ is typically a set of numerical values – such as $\mathbb{N}$ (natural numbers), $\mathbb{Z}$ (integers), $\mathbb{Q}$ (rationals) or $\mathbb{R}$ (reals) – augmented by a not-applicable symbol $\perp$, although our framework only requires that the values and operators define a semi-ring [5].

We assume that all policy operators $\oplus$ are binary and commutative, and can thus be extended unambiguously to k-ary operators for any integer $k > 2$. We further assume that $\perp$ is a *unit* for all these operators (that is $\oplus(\perp, d) = \oplus(d, \perp) = d$) and that $\oplus$ may act as a unary operator (with $\oplus(d) = d$ for all d in $\mathbb{D}$). Operators $\oplus$ will usually be familiar ones, such as addition, multiplication, maximum and minimum, for which the above assumptions are reasonable. To some extent, our definition for the decision set is close to that of the D-algebra [21], since we also consider non restricted sets of decisions with a unit value, however we do not limit ourselves to a particular set of operators.

The decision returned by a policy (its *semantics*) is an element of the decision set. The semantics of a policy are determined by an *evaluation context*, which associates a (binary) truth value with each predicate appearing in the policy. In an access control system, for example, every authorization request provides an evaluation context. We write $[\![\pi, \gamma]\!]$, which belongs to $\{0, 1\}$, to denote the truth value of predicate π in evaluation context γ; and we write $[\![p, \gamma]\!]$, which belongs to $\mathbb{D}$, to denote the value of policy p in evaluation context γ.

Definition 1 *Let $\mathbb{D}$ be a set of decisions,* Ops *a set of operators, and Π be a set of predicates.*

1. *An* atomic policy *has form (π, s), where π is a predicate and s is a decision;*

$$[\![(\pi, s), \gamma]\!] = \begin{cases} s & \text{if } [\![\pi, \gamma]\!] = 1, \\ \perp & \text{otherwise.} \end{cases}$$

2. *If* $p_1, \ldots, p_n$ *are policies and* $\oplus \in \mathsf{Ops}$*, then* $(\oplus, \{p_1, \ldots, p_n\})$ *is a policy;.*

$$[\![(\oplus, \{p_1, \ldots, p_n\}), \gamma]\!] = \bigoplus_{i=1}^{n} [\![p_i, \gamma]\!] .$$

We anticipate that our aggregation languages will be used to support access control decision-making and that policy decisions will provide the link between aggregation policies and authorization policies. In this respect, an aggregation language is similar to the (sub)languages that are used to define "targets" in attribute-based access control languages such as XACML and PTaCL. With this in mind, we introduce the following definition.

Definition 2 *Let* $\mathcal{L} = (\mathbb{D}, \mathsf{Ops}, \Pi)$ *be a policy language. Then a* policy integration rule *(for* $\mathcal{L}$*) is a predicate* ρ *taking elements of* $\mathbb{D}$ *or* $\mathcal{L}$ *as input.*

A policy integration rule (PIR) is also evaluated with respect to a context γ and returns a value in $\{0, 1\}$. If $t \in \mathbb{D}$ and p is a policy, then the PIR $\mathsf{isLessThan}(t, p)$, for example, returns true in context γ iff $t < [\![p, \gamma]\!]$. Henceforth, we will only consider PIRs of the form $\mathsf{isLessThan}(t, p)$, which we will write as $t < p$ for brevity, and $\neg\mathsf{isLessThan}(t, p)$, which we will write as $p \leq t$. PIRs may be "plugged" into all sorts of languages that are used to make decisions, be these decisions about IT systems and their resources, about steps in an off-line workflow such as a mortgage application, etc. A chosen application domain may further constrain our language. For example, if scores model probabilities or likelihoods, we could expect decisions range over the unit interval $[0, 1]$.

In certain situations, we may wish to provide a *default decision* $d \in \mathbb{D}$ for a policy p in the event that all p's sub-policies evaluate to $\perp$ in some context γ, whence $[\![p, \gamma]\!] = \perp$. We can achieve this by including the subpolicy $(\mathbf{1}, d)$ in p, where $\mathbf{1}$ denotes the predicate that always evaluates to true; now $[\![p, \gamma]\!] = d$.

3 Aggregation language

In the remainder of this paper, we focus on a language we have called Peal (Pluggable Evidence Aggregation Language), as a particular instantiation of our policy framework for aggregation. In Peal, the set of decisions is the non-negative real numbers and the operators are $+$, *min* and *max* (although most of our results generalize to more abstract settings). The latter limitation is motivated by pragmatism: we aim to have composition patterns that are at the same time intuitive enough to a policy author, reasonably expressive, yet also amenable to analysis so that we can verify that a policy reflects the intent of its authors.

We use a BNF representation for Peal's syntax, as depicted in Figure 1. Given a predicate q ranging over Π and a non-negative number *score*, a rule (q, s) is written *if* (q) *score*. A policy $p = (op, \{r_1, \ldots, r_n\}, score)$, where each r_i is a rule, *op* is either the addition, *max* or *min* and *score* is a non-negative number, is written as $op(rule^+)$ *default score*. For each policy, we assume that a predicate is listed at most once, i.e., it cannot have two different scores associated within a given policy. However, the same predicate may occur in different policies and be associated with different scores. As 0 is the unit for $+$, we assume that no predicate score in a $+$ policy is 0.

The evidence aggregated in policies can then be combined in expressions *pSet* through applications of *min* and *max*. A *pSet* expression corresponds to a policy set. Peal does not allow the use of the $+$ operator in *pSet* expressions. This deliberate design decision provides a stratification of evidence aggregation into two layers: the lower layer of policies allows each policy to take its

$$
\begin{array}{rcl}
rule & ::= & \mathit{if}\ (q)\ score \\
op & ::= & + \mid min \mid max \\
pol & ::= & op(rule^{+})\ \mathit{default}\ score \\
pSet & ::= & pol \mid max(pSet, pSet) \mid min(pSet, pSet) \\
cond & ::= & th < pSet \mid pSet \leq th
\end{array}
$$

Figure 1: Language Peal for evidence aggregation

$$
\begin{array}{rcl}
b_1 & = & +((\mathit{if}\ (lowCostTransaction\ 0.3)\,(\mathit{if}\ enoughMutualFriends\ 0.1) \\
 & & \quad (\mathit{if}\ enoughMutualFriendsNormalized\ 0.2))\ \mathit{default}\ 0 \\
b_2 & = & min((\mathit{if}\ (highCostTransaction\ 0.1)\,(\mathit{if}\ aFriendOfAliceUnfriendedBob\ 0.2) \\
 & & \quad (\mathit{if}\ aFriendOfAliceVouchesForBob\ 0.6))\ \mathit{default}\ 1 \\
cond & = & 0.5 < min(b_1, b_2)
\end{array}
$$

Figure 2: Expression *cond* models whether Alice is willing to risk paying Bob, based on the amount of payment and on attributes taken from Alice's and Bob's social network.

own "posture" on evidence (e.g. accumulative for +, pessimistic for *min*, etc.); whereas the upper layer combines evidence computed by policies through *min* and *max* only.

A condition expression *cond* is a policy integration rule. Figure 2 shows an example of a PIR in Peal, which models how Alice might want to evaluate the risk of paying Bob, say via PayPal, on a social network such as Facebook. Alice is willing to take that risk if the policy set $min(b_1, b_2)$ has value larger than the threshold 0.5. As *min* is used to compose policies b_1 and b_2, both policies require such a larger value.

Policy b_1 accumulates evidence in support of taking that risk. If the cost of the transaction is sufficiently low, the first rule contributes 0.3 to the value of b_1. Somewhat smaller contributions are made if there are enough mutual friends between Alice and Bob given in two variants discussed further below.

Policy b_2 takes a pessimistic stance and assigns low trust scores when the cost of the transaction is high and when some friend of Alice unfriended Bob. However, if a friend of Alice vouches for the trustworthiness of Bob, a higher trust score is assigned. Since the default score is 1, that last rule is actually redundant for b_2 – something that policy analysis should be able to detect.

In general, scores of rules may be determined in a variety of ways. For example, if predicates are features discovered in machine learning, then scores might be computed probabilities. In our PayPal example, scores could be determined by ranking the importance of rules, as familiar from algorithmic game theory. In policy b_1, Alice gives most support to low-cost transactions, then to a normalized friends-based metric, and then to its unnormalized equivalent. In policy b_2, Alice's biggest risk is represented by a high-cost transaction, followed by a signal in a social network that Alice would interpret as grounds for distrust.

This example also illustrates that predicates in Π may be structured and capture logical dependencies. Figure 3 shows possible definitions for the predicates in *cond* above. Note that $lowCostTransaction$ and $highCostTransaction$ are mutually exclusive but neither logically im-

$$
\begin{aligned}
lostCostTransaction &= (amountAlicePays < 100) \\
highCostTransaction &= (1000 < amountAlicePays) \\
enoughMutualFriends &= (4 < numberOfMutualFriends) \\
enoughMutualFriendsNormalized &= (numberOfBobsFriends < \\
&\qquad 100 * numberOfMutualFriends)
\end{aligned}
$$

Figure 3: Possible definitions of predicates used on the expression *cond* from Figure 2.

plies the negation of the other. So for a transaction amount of 500, say, rules with these predicates have no effect.

Predicate $enoughMutualFriends$ here requires at least five mutual friends, this definition may be subjective and might even include particular friends in a refinement. The predicate $enoughMutualFriendsNormalized$ is a refinement of $enoughMutualFriends$ that shields against scenarios in which Bob has many, many friends (e.g. as for a celebrity).

We note that we could, in principle, encode Peal policies in an attribute-based access control language such as XACML or a Datalog-like authorization language such as SecPal [3]. The first four lines of the policy in Fig. 2 could, for example, be decomposed into many different cases and then encoded as an XACML target. However, this decomposition is likely to be time-consuming and complex (and, therefore, error-prone). Moreover, the entire decomposition may have to be performed whenever an administrator wishes to change one or more of the values in the policy. In the next section, we describe how Peal policies can be automatically transformed into equivalent, purely logical formulae. The resulting formulae could be encoded directly in XACML, say. However, a dedicated engine for evaluating Peal policies will be more efficient in practice. Rather, the value of the transformation is to use off-the-shelf SAT solvers and SMT solvers to *analyze* Peal policies (as we will discuss in Sec. 5).

4 Logical synthesis

Evidence may stem from a variety of sources and ontologies. For example, the evidence that some software is running on a specific platform is very different in character from the information that the software is ten years old or that the machine on which it is running is in a particular legal territory. The aggregation of such evidence in $pSet$, in combination with its comparison in *cond* therefore poses a risk (no pun intended). The choice of scores, comparison operators, and composition operators may result in values for *cond* that are un-anticipated or undesired when *cond* is seen as a basis for making access-control decisions.

Ideally, we would want a tool that allows us to confirm that expressions *cond* meet the expectations of their authors. For example, we may want to assess whether a change to some scores (be it for the thresholds in *cond*, default scores of *pol* or scores of predicates q) will have any or an expected change impact on the truth of conditions *cond*; or we may want to ensure that *cond* is not vacuously true, etc.

4.1 Inductive synthesis process

We show how to synthesize formulae of propositional logic over Π from *cond* expressions, thereby allowing us to apply satisfiability solving to the questions described above. We write $\phi[pSet \leq th]$ for the formula that captures the meaning of expression $pSet \leq th$. Assignments ρ that map predicates q_i to truth values *true* or *false* operate at a higher level of abstraction than evaluation contexts γ.

Definition 3 *Two evaluation contexts γ and γ' are* equivalent *if $[\![\pi, \gamma]\!] = [\![\pi, \gamma']\!]$ for each predicate π in Π. If γ and γ' are equivalent, we say they induce the same truth assignment on Π, and write ρ^γ for the assignment induced by the equivalence class containing γ.*

The aim of our synthesis is that $[\![pSet \leq th, \gamma]\!]$ equals 1 if, and only if, $\phi[pSet \leq th]$ is true under assignment ρ^γ. Expression $pSet \leq th$ is the logical negation of $th < pSet$ as $\leq$ is a total order over non-negative reals. The synthesis for the latter expression thus reduces to that of the former by defining

$$\phi[th < pSet] = \neg\phi[pSet \leq th] \tag{1}$$

The synthesis of aggregation of policy evidence is defined inductively:

$$\begin{aligned} \phi[min(pS_1, pS_2) \leq th] &= \phi[pS_1 \leq th] \vee \phi[pS_2 \leq th] \\ \phi[max(pS_1, pS_2) \leq th] &= \phi[pS_1 \leq th] \wedge \phi[pS_2 \leq th] \end{aligned}$$

Assuming that $\phi[pS_i \leq th]$ correctly captures the operational truth of $pS_i \leq th$ under all truth assignments, these inductive definitions encode the correct meaning of *min* and *max* over the reals: $min(pS_1, pS_2) \leq th$ holds if, and only if, $pS_i \leq th$ holds for some i in $\{1, 2\}$ (a disjunction); and $max(pS_1, pS_2) \leq th$ holds if, and only if, $pS_i \leq th$ holds for all i in $\{1, 2\}$ (a conjunction).

It remains to show how to synthesize $\phi[pol \leq th]$ for policies *pol*, which have form $op(rule^+)$ *default* s. The top-level structure of formulae for *op* policies is always a disjunction: either we consider the default score "*df*" (when no predicates hold) or we consider the aggregate score ("*ndf*"):

$$\phi[pol \leq th] \stackrel{\text{def}}{=} \phi^{df}[pol \leq th] \vee \phi^{ndf}_{op}[pol \leq th] \tag{2}$$

Clearly, $\phi^{df}[pol \leq th]$ is the same for all operators *op*:

$$\phi^{df}[pol \leq th] \stackrel{\text{def}}{=} (s \leq th) \wedge \bigwedge_{i=1}^{n} \neg q_i \tag{3}$$

We interpret $(s \leq th)$ as *true* (a redundant conjunct) when constant s is less than or equal to constant th; otherwise, we interpret $\phi^{df}[pol \leq th]$ as *false* (a redundant disjunct in (2)).

It remains to define the formula $\phi^{ndf}_{op}[pol \leq th]$ for each operator *op* in our language. In doing so, we interpret empty disjunctions as *false* and empty conjunctions as *true*. We model *pol* as a list of predicate-score pairs $[(q_1, s_1), \ldots, (q_n, s_n)]$ and write $[n]$ to denote the set of indices $\{1, \ldots, n\}$.

Let $T \subseteq [n]$ be the non-empty set of indices of predicates that hold in *pol* in context γ. Then $[\![pol, \gamma]\!]$ equals $op(\{s_i \mid i \in T\})$ and $\phi^{ndf}_{op}[pol \leq th]$ should hold if, and only if, $op(\{s_i \mid i \in T\}) \leq th$. We define the set of all such T as

$$\mathsf{Ord}[th, op] \stackrel{\text{def}}{=} \{T \subseteq [n] \mid T \neq \emptyset, op(\{s_i \mid i \in T\}) \leq th\} \tag{4}$$

Thus, $\phi^{ndf}_{op}[pol \leq th]$ should hold if, and only if, there is some T in $\mathsf{Ord}[th, op]$ where T equals the set of predicates that are true in pol. In other words, the formula for the synthesis of $pol \leq th$ with operator op may be defined as:

$$\phi^{ndf}_{op}[pol \leq th] \stackrel{\text{def}}{=} \bigvee_{T \in \mathsf{Ord}[th,op]} (\bigwedge_{j \notin T} \neg q_j \wedge \bigwedge_{j \in T} q_j) \tag{5}$$

We can make this formula more compact when op is monotone with respect to the partial order $\leq$ of the semi-ring, i.e. when $T \subseteq T'$ implies $op(T) \leq op(T')$ in the underlying semi-ring. This is the case for $+$ over non-negative reals, for example. Thus, for each T' in $\mathsf{Ord}[th, op]$ we can capture the scenario that only predicates with index in $T \subseteq T'$ are true by stipulating that all predicates q_j, where j is not in T', be false, giving rise to the conjunct $\bigwedge_{j \notin T'} \neg q_j$. Since all scores in Peal are non-negative, $T \subseteq T'$ and T' in $\mathsf{Ord}[th, op]$ imply T in $\mathsf{Ord}[th, op]$. We order $\mathsf{Ord}[score, op]$ by subset inclusion and write $\mathsf{M}[th, op]$ for the set of maximal elements of $(\mathsf{Ord}[th, op], \subseteq)$. Since every element of $\mathsf{Ord}[th, op]$ is a subset of some element in $\mathsf{M}[th, op]$, we need only consider formulae of form $\bigwedge_{j \notin T'} \neg q_j$ for elements of $\mathsf{M}[th, op]$. Thus, for monotone operators op we have

$$\phi^{ndf}_{op}[pol \leq th] = \bigvee_{T \in \mathsf{M}[th,op]} \bigwedge_{j \notin T} \neg q_j \tag{6}$$

and note that this formula is then logically equivalent to the formula in (5), limits disjunctions to maximal elements, and removes the second conjunct from each such disjunct.

We may also simplify the formula in (5) for some concrete operators such as min and max. For the former, we need that at least one q_i with $s_i \leq th$ be true. For the latter, we need that at least one q_i be true (to force the non-default case) and to have no q_i with $th < s_i$ be true:

$$\begin{aligned} \phi^{ndf}_{min}[pol \leq th] &= \bigvee \{q_i \mid s_i \leq th\} \\ \phi^{ndf}_{max}[pol \leq th] &= \bigvee_{i=1}^{n} q_i \wedge \bigwedge \{\neg q_i \mid th < s_i\} \end{aligned} \tag{7}$$

We state the formal correctness of this synthesis process; the proof of Theorem 1 can be found in Appendix B.

Theorem 1 *Let $pSet$ be an arbitrary expression of* Peal *and γ be an evaluation context. Then:*

1. *Expression $[\![pSet \leq th, \gamma]\!]$ equals 1 iff formula $\phi[pSet \leq th]$ evaluates to true under ρ^γ.*
2. *Expression $[\![th < pSet, \gamma]\!]$ returns 1 for ρ iff formula $\phi[th < pSet]$ evaluates to true under ρ^γ.*

4.2 Expressive power of synthesis

We have seen that it is possible to transform statements in Peal into a logical form; namely, propositional logic over Π. A natural question is then what formulae of propositional logic over Π can be obtained from Peal. It turns out that the answer is *all of propositional logic*. We state the result formally, the proof of which can be found in the appendix.

Theorem 2 *For each expression cond of our language, $\phi[cond]$ is a formula of propositional logic over Π. Conversely, there are scores x, y, and z with $x < y < z$ such that every propositional logic formula over Π is logically equivalent to some $\phi[pSet \leq y]$ that uses only min policies and only the scores x, y, and z.*

4.3 Synthesis algorithm

The definitions of $\phi[pSet \leq th]$ and $\phi[th < pSet]$ make it clear that their implementation is linear in the size of the expression $pSet$ and constant in the size of th, except for the computation of $\phi[pol \leq th]$ (and so of $\phi[th < pol]$ as well) for operators such as + that rely on the computation of $\mathsf{M}[th, op]$, as in (6). For such operators, our synthesis is linear in the size of pol and $\mathsf{M}[th, op]$.

The computation of an element of $\mathsf{M}[th, +]$ can be seen as a special (and simple) instance of the *0-1 knapsack problem*, which seeks to maximize $\sum_{i=1}^{n} t_i p_i$ subject to $\sum_{i=1}^{n} s_i p_i \leq th$ where predicates p_i have value in $\{0, 1\}$ and s_i and t_i are non-negative. This optimization problem is NP-hard. But in our setting, all t_i equal 1 and so we can optimize this efficiently. However, computing set $\mathsf{M}[th, +]$ amounts to enumerating *all* solutions to this easier optimization problem.

In that context, we note that the size of $\mathsf{M}[th, op]$ can be exponential in the number n of predicates of an *op* policy: let *op* be +, n be even, and each of the n predicates q_i have score $1/n$. Let th be 0.5. When *cond* is $0.5 < pol$ this models majority voting, which can be evaluated in linear time. We can synthesize $\phi[0.5 < pol]$ as the logical negation of $\phi[pol \leq 0.5]$. Here, $\mathsf{M}[0.5, +]$ contains all and only those subsets of $[n]$ with exactly $n/2$ elements.

We now present and evaluate an algorithm that seeks to minimize the number of elements of $\mathsf{Ord}[th, +]$ that are explored in that computation. We assume that we have a list of n predicate-score pairs $[(q_1, s_1), \ldots, (q_n, s_n)]$ such that $0 < s_1$ and $s_i \leq s_{i+1}$ for all i. Our algorithm is designed with the following observations in mind. Let I be some subset of $[n]$ and suppose j and k are in I with $j < k$ (hence $s_i \leq s_j$). Then

$$\sum_{i \in I} s_i > \sum_{i \in I \setminus \{j\}} s_i \geq \sum_{i \in I \setminus \{k\}} s_i > \sum_{i \in I \setminus \{j,k\}} s_i.$$

Now suppose that $\sum_{i \in I} s_i \not\leq t$.

- If $\sum_{i \in I \setminus \{j\}} s_i \leq t$, then $\sum_{i \in I \setminus \{k\}} \leq t$ also.
- If $\sum_{i \in I \setminus \{j\}} s_i \not\leq t$ and $\sum_{i \in I \setminus \{k\}} \leq t$, then we do not need to consider $\sum_{i \in I \setminus \{j,k\}} s_i$.
- If $\sum_{i \in I \setminus \{j\}} s_i \not\leq t$ and $\sum_{i \in I \setminus \{k\}} \not\leq t$, we need to consider $\sum_{i \in I \setminus \{j,k\}} s_i$ only once (when we omit k first).

The algorithm `enum_sring` for the computation of $\mathsf{M}[th, +]$ is depicted in Figure 4. The algorithm terminates a search branch as soon as a score under the threshold is found; otherwise, it recursively starts new branches implementing the strategy determined by the observations above.

```
enum_sring (S, th, i)
  if ∑S ≤ th then
    output indices of S with non-zero scores
  else
    for j = i down to 1
      enum_sring (S{j ↦ 0}, th, j − 1)
```

Figure 4: Recursive algorithm `enum_sring` for enumerating all elements of $\mathsf{M}[th, +]$ where the array $S\{j \mapsto 0\}$ is S except at position j, which stores 0

For the array of scores $[0.1, 0.2, 0.2, 0.3]$, and a threshold $th = 0.5$, the call `enum_sring`$([0.1, 0.2, 0.2, 0.3], 0.5, 4)$ results in the following trace (indentation indicating recursive calls).

```
enum_sring ([0.1,0.2,0.2,0.3],0.5,4)
    enum_sring ([0.1,0.2,0.2,0],0.5,3) -> output [1,2,3]
    enum_sring ([0.1,0.2,0,0.3],0.5,2)
        enum_sring ([0.1,0,0,0.3],0.5,1) -> output [1,4]
        enum_sring ([0,0.2,0,0.3],0.5,0) -> output [2,4]
    enum_sring ([0.1,0,0.2,0.3],0.5,1)
        enum_sring ([0,0,0.2,0.3],0.5,0) -> output [3,4]
    enum_sring ([0,0.2,0.2,0.3],0.5,0)
```

Figure 5 shows average execution times of `enum_sring` to compute $\mathsf{M}[th, +]$ for policies containing up to 30 rules. For each n, $1 \leq n \leq 30$, we generate 20 policies, each containing n predicates with scores $s_1 \leq \cdots \leq s_n$, with $s_1 = 1$ and, for all $i > 1$, s_{i+1} is chosen at random from the range $[s_i, s_i + 4]$. We define th to be a random value in the interval $[0, \sum_{i=1}^{n} s_i]$. These results confirm an exponential growth in the running time with respect to the number of rules and have been obtained with the iterative implementation of the `enum` algorithm in Ocaml [19], using native-code compilation (ocamlopt), on a 2 GHz Intel core i7 with 8GB of RAM.

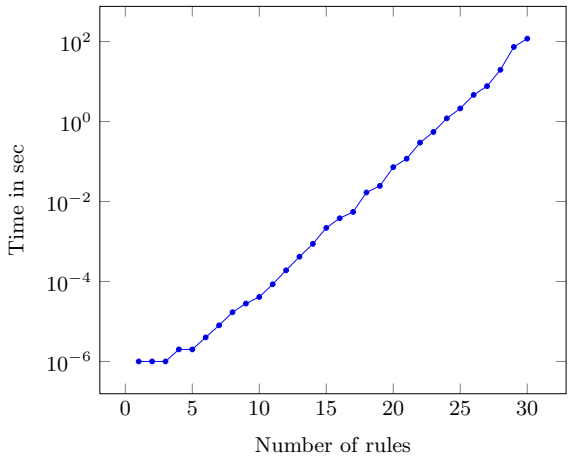

Figure 5: Time taken to compute $\mathsf{M}[th, +]$ using `enum_sring`

Figure 6 shows the number of elements visited by `enum_sring` as a function of the number of predicates, using the randomly generated policies described above. This value cannot be larger than 2^n for policies with n rules. For a few policies, all or almost all elements were visited. But since the y-axis uses a logarithmic scale, `enum_sring` performs considerably better than exhaustive search on most of these problem instances.

Finally, Figure 7 presents the time required to compute $\mathsf{M}[th, +]$ for a policy of 25 rules where each rule score is 1 and th varies from 1 to 25, thus implementing majority-voting policies. We can clearly see that thresholds close to half the number of rules have worst performance (as expected). The graph is asymmetric since `enum_sring` starts by considering the set of all scores and removes elements. (The asymmetry would be reversed if it were to start with empty score set.)

4.4 From semi-rings to rings

Thus far, we have advocated the use of semi-rings for the aggregation of evidence, where different semi-rings may be used in different policies. But there are settings where the interaction of

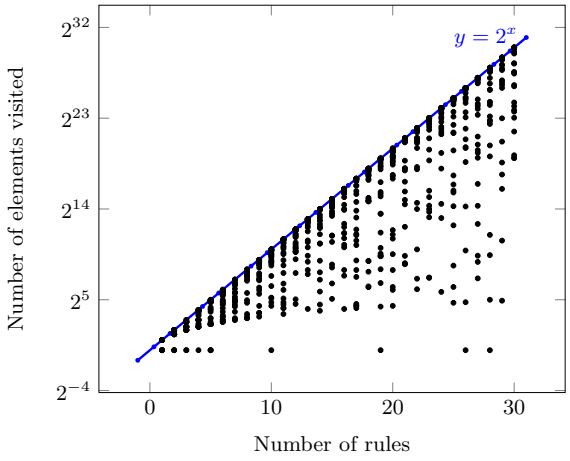

Figure 6: Number of elements visited by `enum_sring`

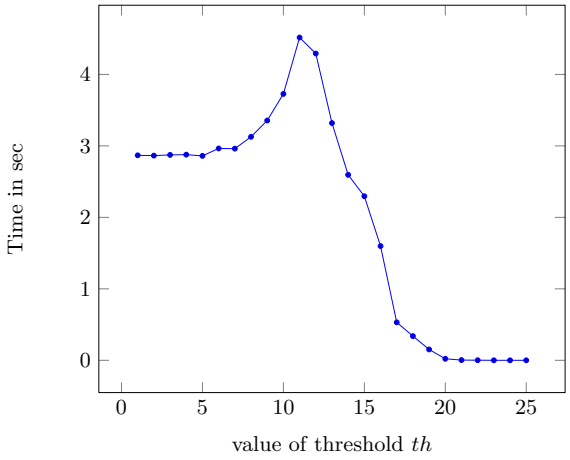

Figure 7: Computation time for `enum_sring` on a 25-rule majority-voting policy

positive and negative evidence is important, for example when considering arguments and counter arguments. We therefore want to discuss how our approach adapts to the setting of *rings*, where the monoid $(A, +, 0)$ is actually a group with inverse $a \mapsto -a \colon A \to A$.

Language Peal is unaffected by such a change. The logical synthesis process described above is unaffected as well, with the exception of the formula in (6). We now illustrate how this formula would change for the ring of reals and *op* being $+$. Set $\mathsf{Ord}[th, +]$ is as in (4). For X in $\mathsf{Ord}[th, +]$, define

$$X_p = \{i \in X \mid s_i > 0\} \qquad X_n = \{i \in X \mid s_i < 0\}$$

The intuition is that since $\sum_{i \in X} s_i \leq th$, we may remove indices from X_p or add indices with negative score to X_n as long as this results in a non-empty set. Therefore, we order $\mathsf{Ord}[th, +]$ now as $X \sqsubseteq Y$ iff $X_p \subseteq Y_p$ and $Y_n \subseteq X_n$. Under this ordering, summation is monotone: $X \sqsubseteq Y$

implies $\sum_{i\in X} s_i \leq \sum_{i\in Y} s_i$. The formula in (6) becomes

$$\phi_+^{ndf}[rule \leq th] = \bigvee_{X\in \mathsf{M}[th,+]} \bigwedge_{i\in X_n} q_i \wedge \bigwedge_{i\notin X_p, s_i>0} \neg q_i, \tag{8}$$

where $\mathsf{M}[th,+]$ is now the set of maximal elements in $(\mathsf{Ord}[th,+],\sqsubseteq)$. The other change concerns the shape of the disjunct for X: it is still a conjunction, but now lists predicates with negative score whose index is in X_n, and negated predicates with positive score whose index is not in X_p. We now describe how `enum_sring` needs to be modified to accommodate negative scores as well.

Given a list $[(q_1,s_1),\ldots,(q_n,s_n)]$ and an index set $I\subseteq[n]$, we may reduce $\sum_{i\in I} s_i$ by removing a positive score or by including a negative score. We assume an array of scores S sorted in ascending order of the absolute values of scores. That is, $|S[1]| \leq |S[2]| \leq \ldots \leq |S[n]|$. We define an array B, where $B[i]$ encodes the index set I: that is $B[i]=1$ if and only if $i\in I$. Then

$$\sum_{i\in I} s_i = \sum_{i=1}^{n} B[i] * S[i]$$

We may now use a similar backtracking algorithm to the one specified in Figure 4, as shown in Figure 8. The algorithm is called with $i=n$ and $B[j]=1$ if and only if $s_j>0$. (In other words, we start by computing the sum of all positive scores.) An example run of the program is given in Appendix A.

```
enum_ring (S, B, t, i)
  if ∑_{i=1}^{n} S[i] * B[i] ≤ t then
    output indices with non-zero scores
  else
    for j = i down to 1
      B[j] + 1 mod 2 ← B[j]   /* flip bits */
      enum_ring (S, B, t, j − 1)
```

Figure 8: Enumerating all elements of $\mathsf{M}[th,+]$ when scores may be positive or negative

5 Analyzing Policies

We show how to reduce the analysis of *cond* expressions to reasoning about the synthesis of such conditions, thereby reducing questions of policy analysis to logical reasoning. Our discussion focuses on conditions of the form $pSet \leq th$ for the sake of concreteness. The discussion for $th < pSet$ is very similar.

5.1 Vacuity and redundancy analysis

The first problem we discuss is *vacuity analysis*, which is valuable in the specification and verification of hardware design (see [1, 18], for example). In our context, it serves to identify expressions that always evaluate to the same truth value and is useful because it is unlikely that we would want to base a decision on a condition whose evaluation is true in all contexts (or always false, for that matter). By appeal to Theorem 1, we can establish that $pSet \leq th$ is always true by checking that

formula $\phi[pSet \leq th]$ is logically valid; and establishing the satisfiability of $\phi[pSet \leq th]$ means that $pSet \leq th$ is not always false. But vacuity of certain logical statements may also be a good thing in our setting: we may want to ensure that the evidence computed in a policy is always positive, for example. This we can do by verifying that $pol \leq 0$ is always false.

Given a policy or policy set written in Peal, there is the possibility that some predicate within a policy does not really contribute anything and its inclusion therefore introduces unnecessary computation during policy evaluation. Thus, *redundancy analysis* is important and could be interpreted in two ways: no contribution in the evaluation of the policy, or no contribution in the evaluation of an expression *cond* that refers to this policy. We consider the second case and write $cond \setminus q$ to denote the expression obtained from *cond* be removing all rules that involve q from all policies in *cond*. This assumes that no policy contains a single rule with predicate q. We can then check whether $\phi[cond] \leftrightarrow \phi[cond \setminus q]$ is logically valid. If so, the presence of q makes no difference to the policy and this might suggest a specification error.

5.2 Sensitivity analysis

An important analysis is that of the *sensitivity* of the value of th in the evaluation of conditions $pSet \leq th$. Given $th < th'$, a satisfiability witness of the formula

$$\phi[pSet \leq th'] \wedge \neg\phi[pSet \leq th], \tag{9}$$

may explain why an increase from a score of th to th' flips the truth value of $\phi[pSet \leq th]$ from *false* to *true* in the same evaluation context.

We may take this kind of analysis further. Suppose, for example, that we are using the unit interval $[0, 1]$ for the set of decisions. We may wish to establish the "tipping points" for the threshold score, at which truth values change. If we wanted to set thresholds at intervals of 0.2, for example, we could then verify (or otherwise) that the increase from 0 to, say, 0.19 does not change any meaning.

In a more complex application, we may have an interval $[l, u]$ of possible values for th and use satisfiability solving in combination with binary search to find closed subintervals in $[l, u]$ in which the evaluation of $pSet \leq th$ is fixed for any choice of th from a given subinterval. Initially, we set th and th' to l and u, respectively and evaluate equation (9). We then repeat for the intervals $[l, l + (u - l)/2]$ and $[l + (u - l)/2, u]$, and so on, until the desired level of granularity has been achieved. This would certainly further our understanding of the relevance of threshold values and would strengthen our confidence in using them. In fact, it may be used to certify that a policy does implement the right postures for a fixed set of discrete threat levels th.

We may also investigate the effect of including a particular policy *pol* in a *cond* expression. For the policies $pSet$ and $min(pol, pSet)$, e.g., the formulae

$$\phi[pSet \leq th] \wedge \neg\phi[min(pol, pSet) \leq th] \tag{10}$$

$$\phi[min(pol, pSet) \leq th] \wedge \neg\phi[pSet \leq th] \tag{11}$$

have as satisfiability witnesses all those scenarios in which the inclusion of policy *pol* turns *false* into *true* (for the first formula above) or *true* into *false* (for the latter formula). An application of this might be to ensure that the inclusion of a *best* policy *pol* (which takes an optimistic view on things with low risk aversion) does not undermine important concerns expressed in $pSet$.

It is potentially attractive to do sensitivity analysis without using satisfiability solvers. For example, if a change from th to th' in (9) meant that the synthesized formulae $\phi[pSet \leq th]$ and

$\phi[pSet \leq th']$ have the same concrete syntax, then it is clear that this change of score results in equivalent conditions $pSet \leq th$ and $pSet \leq th'$. We illustrate this on the shape of formula $\phi[pol \leq th]$ for a *min* policy. For formula (2), we just require that the truth value of $s \leq th$ remains unchanged when we want to change either s or th. But we also need to make sure that the formula in (7) won't change syntactically. This means we can manipulate th and the set of predicate scores s_i in any way we choose, as long as it still returns the same set of predicates for which $s_i \leq th$ is true. This syntactic method seems less suited for + policies, though, as one would have to ensure that the set of maximal elements $\mathsf{M}[th]$ remains the same.

5.3 Certifying that policies are safe

Attribute-based access control is particularly useful in open and distributed systems, where management of security information might need to be decentralized. Attribute values are then collected from different sources, including the user herself, which raises the question of policy behavior when information is withheld, intentionally or not [12].

According to Tschantz and Krishnamurthi, a policy is *safe* if "incomplete requests should only result in a grant of access if the complete one would have" [25]. In the context of attribute-based access control, a predicate may evaluate to *true* if a given attribute value is present in the request, and to *false* otherwise. In other words, an incomplete request can be seen as a request that satisfies a subset of the predicates satisfied by the complete request.

An attacker can choose different partial requests to attack different sets of predicates, including the empty set of predicates [15]. We therefore need the ability to model that choice of an attacker. Given a policy set $pSet$, let us write q_i' for a fresh copy of predicate q_i occurring in $pSet$. The intuition is that q_i' either equals q_i (when this predicate is not affected by withholding attribute information) or that q_i' equals *false* (when certain information is withheld that makes q_i false). Therefore, we would expect that each q_i' logically implies q_i whenever the condition $pSet \leq th$ is true within a policy of form *grant if* $(pSet \leq th)$ from a language that includes Peal conditions.

Let us write $\phi\{q_i'/q_i\}$ for the formula ϕ obtained by syntactically substituting each q_i with q_i'. We claim that this policy is safe if:

$$\left(\phi[pSet \leq th]\{q_i'/q_i\} \wedge \bigwedge(q_i' \rightarrow q_i)\right) \rightarrow \phi[pSet \leq th] \tag{12}$$

is logically valid. Therefore, the safety problem for a policy can be reduced to a satisfiability problem over a formula build out of the synthesis of the policy.

Under some assumptions, we can argue formulae in (12) are always valid, meaning that we do not have to check for their validity with any tools. For example, let $pSet = (op, \{(q_1, s_1), \ldots, (q_n, s_n))\}, s)$ such that op is monotonically decreasing (in that $X \subseteq Y$ implies $op(Y) \leq op(X)$), and let us assume that $\phi[pSet \leq th]\{q_i'/q_i\} \wedge \bigwedge(q_i' \rightarrow q_i)$ holds. We need to show that $\phi[pSet \leq th]$ holds under the same assignment of truth values. We first need to assume that the default value s is consistent with op, that is, $s_i \leq s$, for any i. Hence, since op is monotonically decreasing, if the default value satisfies the threshold, then so does any aggregation of scores. Furthermore, by (5), if $\phi_{op}^{ndf}[pSet \leq th]\{q_i'/q_i\}$ holds, then there exists X in $\mathsf{Ord}[th, op]$ such that $\bigwedge_{i \in X} q_i'$ holds. It follows then that $\bigwedge_{i \in X} q_i$ also holds. Since op is monotonically decreasing, any superset of X belongs to $\mathsf{Ord}[th, op]$, and therefore we can conclude that $\phi_{op}^{ndf}[pSet \leq th]$ also holds. In other words, a policy of the form *grant if* $(op, \{(q_1, s_1), \ldots, (q_n, s_n))\}, s) \leq th$ is safe if op is monotonically decreasing and s is consistent with op. For instance, *min* is monotonically decreasing (removing values in X increases the value of $min(X)$), and the policy *grant if* $b_2 \leq 0.5$, where b_2 is defined as in Figure 2, is safe.

Dually, a policy of the form *deny if* $pSet \leq th$ is safe if:

$$\Big(\phi[pSet \leq th] \wedge \bigwedge(q_i \rightarrow q_i')\Big) \rightarrow \phi[pSet \leq th]\{q_i'/q_i\} \tag{13}$$

is logically valid. Note that this definition keeps the meaning of q_i' and q_i used in (12) but now swaps the order of primed and un-primed policies and predicates in the shape of the formula. Indeed, the notion of safe means that a *grant* cannot be obtained by removing information, and a *deny* cannot be erased by adding information. In that case, we can similarly show that if $pSet$ is built only with monotonically increasing operators, the formula in (13) is always logically valid and so the policy is then safe.

6 Related and future work

There has been considerable interest in access control languages and policy algebras that, conceptually, represent policies as trees [7, 8, 12, 23, 26]. The evaluation of a policy with respect to an authorization request uses a post-order traversal of the tree, assigns decisions to the leaf nodes, and then computes decisions for non-leaf nodes using decision-combining operators. In Section 3, we showed how to represent the common decision-combining operators using semi-ring operators over carrier set $\{0, 1\}$. Thus, we believe our approach subsumes such languages and algebras, at least with regard to basic policy specification and evaluation.

Recent work has considered more complex decision-combining operators, the complexity arising either from the choice of decision set or from the operators themselves. Li *et al.* [20], for example, define new operators using linear constraints, which, informally, can compute a decision for a policy based on the number of instances of particular decisions that arise in the evaluation of its sub-policies. A typical example, cited by Li *et al.*, is to return a decision d if more than half the sub-policies evaluate to d. We have seen how the + operator and *cond* expressions can be used to achieve a similar effect. However, our work only considers two possible outcomes for the evaluation of expressions $pSet \leq th$ and $th < pSet$ – whereas Li *et al.* work with a more complex decision set. We hope to extend our *cond* expressions to such decision sets in the future.

There is also a large body of work on risk, trust and reputation, and combining scores that represent these concepts. The survey of Jøsang *et al.* [17] provides a comprehensive introduction to this topic. The focus of such work has tended to be on finding ways of combining scores "transitively": if A trusts B and B trusts C, how much should A trust C? Our focus is on using numerical values (that may represent risk, trust, reputation or any other form of evidence that might inform access control decision-making) to construct policies. There is some work on risk- or trust-aware access control, but its focus has not been on evidence-based policy languages. The work of Chen and Crampton [10], for example, considers an extended decision set and simply describes how a given risk value leads to a particular decision; how risk is computed or aggregated from evidence is not considered. The work of Chakraborty and Ray on trust-aware role-based access control [9] focuses on how to compute a trust value for each user, with those trust values being associated with roles in a role-based access control framework. Our work provides a more structured and flexible framework for aggregating and incorporating trust (or similar concepts that map to some numerical domain) in access control policies. The work of Ni *et al.* [22] considers fuzzy values for security levels, and uses t-norms to aggregate fuzzy values.

The notion of semi-ring has been used to model different aspects of security systems, for instance by providing a trust metric in order to extend the RT language [6]. Semi-rings are also used by

Schwoon *et al.* [24] to compute the weight of a chain of credentials. Non-numerical semi-rings are considered by Bharadwaj and Baras [4], where the authors address the problem of policy negotiation by defining semi-rings for roles. We do not know any prior work that uses semi-rings to aggregate evidence within structured policies.

Resource infrastructures and the demands put on them may be subject to frequent change. Policies that regulate access to such resources may therefore be inconsistent or may not capture important access scenarios (so called "policy gaps"). The work of Fisler *et al.* [13] developed techniques for the verification of RBAC policies and for the detection of semantic differences between two RBAC policies. Inconsistency, gap, and policy refinement analyses were proposed by Bruns and Huth [8] for a policy composition language with explicit decisions for inconsistencies and gaps. Finally, Basile *et al.* [2] provide methods for the detection of inconsistencies and other anomalies in network policies, and policies are translated into rule-based form for efficient processing.

We have shown that it is possible to analyze Peal policies and to identify important properties of such policies in doing so. However, Peal policies have a particular structure and *cond* expressions only return binary decisions. In particular, Peal policies do not give rise to gaps or inconsistencies in their current form. In future work, we want to better understand what types of inconsistencies and gaps may occur in Peal when we make use of richer decision types, and to develop static and dynamic techniques for their detection and resolution. It would also be of interest to derive normalform results for policies in Peal that can facilitate static analysis or minimize run-time overheads. The logical synthesis developed in this paper reduces analyses to satisfiability checks for propositional logic over Π. Since predicates in Π may have logical dependencies and be subject to axioms, SMT solvers seem an obvious choice for implementing such satisfiability checks. In [16], a tool is developed and evaluated that compiles *cond* expressions into analyzable input code of the SMT solver Z3, with partial support for non-constant scores. Optimizations for synthesis of $+$ policies (such as binary decision diagrams and exploitations of symmetry) is subject of future work. Peal will benefit from the analyzable administration of edit permissions on scores and thresholds. Such tool support exists, for example, for administrative RBAC [14]. Finally, we want to investigate how our analysis techniques could be applied to real-world languages such as XACML.

7 Conclusions

In this paper we proposed a new framework for developing authorization policy languages that aggregate evidence for access-control decisions and where this evidence may be quantitative in nature. The novel feature of our framework is that it explicitly incorporates values taken from some semi-ring used to represent security-relevant information such as risk, trust or reputation – and supplies operators for combining those values. We then introduced an instance of our framework, Peal where evidence ranges over non-negative reals, its aggregation happens in two layers (first, security postures aggregate sub-values that are then composed to consider worst-case outcomes), and composition operators model accumulative, pessimistic, and optimistic evidence aggregation respectively. We then demonstrated that the meaning of these numerical policies can be precisely captured in synthesized formulae of a suitable propositional logic, and we developed and evaluated algorithms that support that synthesis process. Applications of that synthesis process were specified by showing how important analysis tasks, such as the sensitivity analysis of scores used in numerical policies, reduce to satisfiability checking of the synthesized formulae.

References

[1] R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Y. Vardi. Enhanced vacuity detection in linear temporal logic. In W. A. H. Jr. and F. Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 368–380. Springer, 2003.

[2] C. Basile, A. Cappadonia, and A. Lioy. Network-level access control policy analysis and transformation. *IEEE/ACM Trans. Netw.*, 20(4):985–998, 2012.

[3] M. Y. Becker, C. Fournet, and A. D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

[4] V. G. Bharadwaj and J. S. Baras. Towards automated negotiation of access control policies. In *POLICY*, pages 111–119. IEEE Computer Society, 2003.

[5] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *LNCS*. SpringerVerlag, 2004.

[6] S. Bistarelli, F. Martinelli, and F. Santini. A semantic foundation for trust management languages with weights: An application to the RT family. In C. Rong, M. G. Jaatun, F. E. Sandnes, L. T. Yang, and J. Ma, editors, *ATC*, volume 5060 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2008.

[7] P. Bonatti, S. De Capitani Di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security*, 5(1):1–35, 2002.

[8] G. Bruns and M. Huth. Access control via Belnap logic: Intuitive, expressive, and analyzable policy composition. *ACM Transactions on Information and System Security*, 14(1):9, 2011.

[9] S. Chakraborty and I. Ray. TrustBAC: integrating trust relationships into the RBAC model for access control in open systems. In D. F. Ferraiolo and I. Ray, editors, *SACMAT*, pages 49–58. ACM, 2006.

[10] L. Chen and J. Crampton. Risk-aware role-based access control. In C. Meadows and M. C. F. Gago, editors, *STM*, volume 7170 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2011.

[11] P.-C. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *IEEE Symp. on Security and Privacy*, pages 222–230. IEEE Computer Society, 2007.

[12] J. Crampton and C. Morisset. PTaCL: A language for attribute-based access control in open systems. In P. Degano and J. D. Guttman, editors, *POST*, volume 7215 of *Lecture Notes in Computer Science*, pages 390–409. Springer, 2012.

[13] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In G.-C. Roman, W. G. Griswold, and B. Nuseibeh, editors, *ICSE*, pages 196–205. ACM, 2005.

[14] M. I. Gofman, R. Luo, A. C. Solomon, Y. Zhang, P. Yang, and S. D. Stoller. RBAC-PAT: A policy analysis tool for role based access control. In *TACAS*, pages 46–49, 2009.

[15] A. Griesmayer and C. Morisset. Automated certification of authorisation policy resistance. In J. Crampton, S. Jajodia, and K. Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 574–591. Springer, 2013.

[16] M. Huth and J. Kuo. PEALT: A reasoning tool for numerical aggregation of trust evidence. Technical Report 2013/7, Imperial College London, Department of Computing, October 2013. ISSN 1469-4166 (Print), ISSN 1469-4174 (Online).

[17] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision support systems*, 43(2):618–644, 2007.

[18] O. Kupferman and M. Y. Vardi. Vacuity detection in temporal model checking. In L. Pierre and T. Kropf, editors, *CHARME*, volume 1703 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 1999.

[19] X. Leroy, D. Doligez, A. Frisch, J. Garrigue, D. Rémy, and J. Vouillon. The OCaml system 4.00, 2012. available at `http://caml.inria.fr/`.

[20] 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, *SACMAT*, pages 135–144. ACM, 2009.

[21] Q. Ni, E. Bertino, and J. Lobo. D-algebra for composing access control policy decisions. In *Proc. of 4th Int'l Symp. on Information, Computer, and Communications Security*, ASIACCS '09, pages 298–309, New York, NY, USA, 2009. ACM.

[22] Q. Ni, E. Bertino, and J. Lobo. Risk-based access control systems built on fuzzy inferences. In *Proc. of 5th ACM Symp. on Information, Computer and Communications Security*, ASIACCS '10, pages 250–260, New York, NY, USA, 2010. ACM.

[23] OASIS. *eXtensible Access Control Markup Language (XACML) Version 3.0*, 2010. Committee Specification 01.

[24] S. Schwoon, S. Jha, T. W. Reps, and S. G. Stubblebine. On generalized authorization problems. In *CSFW*, pages 202–218. IEEE Computer Society, 2003.

[25] M. Tschantz and S. Krishnamurthi. Towards reasonability properties for access-control policy languages. In D. Ferraiolo and I. Ray, editors, *SACMAT 2006,11th ACM Symposium on Access Control Models and Technologies, Proceedings*, pages 160–169. ACM, 2006.

[26] D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. *ACM Transactions on Information and System Security*, 6(2):286–235, 2003.

A Examples

We provide an illustrative run of the algorithm shown in Figure 8, which computes maximal index sets in the presence of positive and negative scores. Let $S = [-0.1, -0.1, 0.2, -0.3, 0.3]$, so B is initialized to $[0, 0, 1, 0, 1]$. The call to `enum_ring` has the following "trace" (S and t are omitted as they do not change), where, for the sake of compactness, we abbreviate `enum_ring` as `enum`.

```
enum([0,0,1,0,1],5)  (score = 0.5)
  enum([0,0,1,0,0],4)  (score = 0.2)
    enum([0,0,1,1,0],3)  (score = -0.1, output [3,4])
    enum([0,0,0,0,0],2)  (empty set, ignore)
    enum([0,1,1,0,0],1)  (score = 0.1, output [2,3])
    enum([1,0,1,0,0],0)  (score = 0.1, output [1,3])
  enum([0,0,1,1,1],3)  (score = 0.2)
    enum([0,0,0,1,1],2)  (score = 0, output [4,5])
    enum([0,1,1,1,1],1)  (score = 0.1, output [2,3,4,5])
    enum([1,0,1,1,1],0)  (score = 0.1, output [1,3,4,5])
  enum([0,0,0,0,1],2)  (score = 0.3)
    enum([0,1,0,0,1],1) (score = 0.2)
      enum([1,1,0,0,1],0) (score =  0.1, output [1,2,5])
    enum([1,0,0,0,1],0) (score = 0.2)
  enum([0,1,1,0,1],1)  (score = 0.4)
    enum([1,1,1,0,1],0) (score = 0.3)
  enum([1,0,1,0,1],0)  (score = 0.4)
```

Note, for example, that $[2, 3] \not\leq [2, 3, 4, 5]$, because the negative index sets are $N = \{2\}$ and $N' = \{2, 4\}$ respectively, while the positive index sets are $P = \{3\}$ and $P' = \{3, 5\}$. Thus $(P, N) \not\leq (P', N')$. Note also that, for either set, the inclusion of an index for any positive score would increase the total to something greater than 0.1, as would the exclusion of an index for any negative score. Thus, both sets are maximal.

B Proofs

Proof of Theorem 1:

We proceed by induction over the structure of $pSet$.

1($\Rightarrow$) Let us assume that $[\![pSet \leq th, \gamma]\!]$ equals 1, and let us show that $\phi[pSet \leq th]$ evaluates to *true* under ρ^γ. First of all, from $[\![pSet \leq th, \gamma]\!] = 1$, we can deduce that $[\![pSet, \gamma]\!] \leq th$. Three cases are then possible:

- $pSet = max(pSet_1, pSet_2)$; In that case, both $max([\![pSet_1, \gamma]\!], [\![pSet_2, \gamma]\!]) \leq th$ and $\phi[pSet \leq th] = \phi[pSet_1 \leq th] \wedge \phi[pSet_2 \leq th]$ hold; From the former, it follows that $[\![pSet_1, \gamma]\!] \leq th$ and $[\![pSet_2, \gamma]\!] \leq th$, and by induction hypothesis, we have that both $\phi[pSet_1 \leq th]$ and $\phi[pSet_2 \leq th]$ evaluate to *true*, thus allowing us to conclude.
- $pSet = min(pSet_1, pSet_2)$; In that case, both $min([\![pSet_1, \gamma]\!], [\![pSet_2, \gamma]\!]) \leq th$ and $\phi[pSet \leq th] = \phi[pSet_1 \leq th] \vee \phi[pSet_2 \leq th]$ hold; From the former, it follows that either $[\![pSet_1, \gamma]\!] \leq th$ or $[\![pSet_2, \gamma]\!] \leq th$, and by induction hypothesis, we have that either $\phi[pSet_1 \leq th]$ or $\phi[pSet_2 \leq th]$ evaluates to *true*, thus allowing us to conclude.

- $pSet = (op, \{(q_1, s_1), \ldots, (q_n, s_n)\}, s)$; In that case, it is enough to show that either $\phi^{df}[pSet \leq th]$ or $\phi^{ndf}_{op}[pSet \leq th]$ holds. Two sub-cases are possible:
  * Either $\rho^\gamma(q_i) = \mathit{false}$ for all $1 \leq i \leq n$, in that case, $[\![pSet, \gamma]\!] = s \leq th$, and it follows that $\phi^{df}[pSet \leq th]$ holds, and we can conclude.
  * Or there exists at least one i such that $\rho^\gamma(q_i) = \mathit{true}$, in which case let T be the set of all i such that $\rho^\gamma(q_i) = \mathit{true}$. By definition, we have $[\![pSet, \gamma]\!] = op\{s_i \mid i \in T\} \leq th$, since $[\![q_j, \gamma]\!] = \bot$ when $\rho^\gamma(q_j) = \mathit{false}$. It follows that $T \in \mathsf{Ord}[th, op]$, and since by construction of T, $\rho^\gamma(q_j) = \mathit{false}$ for $j \notin T$, and $\rho^\gamma(q_j) = \mathit{true}$ for $j \in T$, we can conclude that $\phi^{ndf}_{op}[pSet \leq th]$, as given in Equation (5), holds.

1($\Leftarrow$) Let us assume that $\phi[pSet \leq th]$ evaluates to *true* under ρ^γ, and let us show that $[\![pSet \leq th, \gamma]\!]$ equals 1, i.e., that $[\![pSet, \gamma]\!] \leq th$. Here again, three cases are possible:

- $pSet = max(pSet_1, pSet_2)$; In that case, we have that both $\phi[pSet_1 \leq th]$ and $\phi[pSet_2 \leq th]$ hold. By induction, it follows that both $[\![pSet_1, \gamma]\!] \leq th$ and $[\![pSet_2, \gamma]\!] \leq th$, and therefore that $max([\![pSet_1, \gamma]\!], [\![pSet_2, \gamma]\!]) \leq th$, which allows us to conclude.
- $pSet = min(pSet_1, pSet_2)$; In that case, we have that either $\phi[pSet_1 \leq th]$ or $\phi[pSet_2 \leq th]$ holds. By induction, it follows that either $[\![pSet_1, \gamma]\!] \leq th$ or $[\![pSet_2, \gamma]\!] \leq th$ holds, and therefore that $min([\![pSet_1, \gamma]\!], [\![pSet_2, \gamma]\!]) \leq th$, which allows us to conclude.
- $pSet = (op, \{(q_1, s_1), \ldots, (q_n, s_n)\}, s)$; In that case, we know that either $\phi^{df}[pSet \leq th]$ or $\phi^{ndf}_{op}[pSet \leq th]$ holds.
  * If $\phi^{df}[pSet \leq th]$ holds, then all $\rho^\gamma(q_i) = \mathit{false}$ for all $1 \leq i \leq n$, and $s \leq th$. From the former, we can deduce that $[\![pSet, \gamma]\!] = s$, and therefore we can conclude.
  * If $\phi^{ndf}_{op}[pSet \leq th]$ holds, then there exists at least one $T \in \mathsf{Ord}[th, op]$ such that any $\rho^\gamma(q_j) = \mathit{false}$ for $j \notin T$, and $\rho^\gamma(q_j) = \mathit{true}$ for $j \in T$. Since $T \neq \emptyset$, it is easy to see that $[\![pSet, \gamma]\!] = op\{s_i \mid i \in T\}$, and by definition of $\mathsf{Ord}[th, op]$, we can conclude that $[\![pSet, \gamma]\!] \leq th$.

2 Peal's order relation is the standard comparison over real numbers, and is therefore a total order. Hence, $[\![th < pSet, \gamma]\!] = 1$ is equivalent to $th < [\![pSet, \gamma]\!]$ which is equivalent to $\neg([\![pSet, \gamma]\!] \leq th)$. Moreover, by definition, $\phi[th < pSet] = \neg\phi[pSet \leq th]$, and we can conclude from step 1. □

Proof of Theorem 2:

We only have to prove the second claim. We use structural induction to prove that all formulas of propositional logic over Π are logically equivalent to some $\phi[pSet \leq 0.1]$. Without loss of generality, we may assume that the propositional formulas over Π are in negation normal form. For sake of concreteness, we choose $x = 0.05$, $y = 0.1$, and $z = 1$ below.

1. We first show that claim for all literals over Π.

   Let q be in Π. Consider the expressions

   $$(min(\mathit{if}\ (q)\ 1)\ \mathit{default}\ 0.05) \ \leq\ 0.1 \tag{14}$$
   $$(min(\mathit{if}\ (q)\ 0.05)\ \mathit{default}\ 1) \ \leq\ 0.1 \tag{15}$$

   of form $pol \leq 0.1$. The corresponding synthesized formula for (14) equals the disjunction of $\neg q$ (as $0 \leq 0.1$ is true) with $\bigvee \emptyset = \mathit{false}$ (as $1 \not\leq 0.1$). But $\neg q \vee \mathit{false}$ is logically equivalent to $\neg q$. The synthesized formula for (15) equals the disjunction of *false* (as $1 \leq 0.1$ is false) with $\bigvee q = q$; that disjunction is logically equivalent with q.

2. Next, we show that claim for disjunction and conjunction. So let ψ_1 and ψ_2 be formulas of propositional logic in negation normal form over Π. By induction hypothesis, there are policy sets $pSet_1$ and $pSet_2$ such that ψ_i is logically equivalent to $\phi[pSet_i \leq 0.1]$ for $i = 1, 2$. But then $\phi[max(pSet_1, pSet_2) \leq 0.1]$ is logically equivalent to the conjunction $\psi_1 \wedge \psi_2$ whereas $\phi[min(pSet_1, pSet_2) \leq 0.1]$ is logically equivalent to the disjunction $\psi_1 \vee \psi_2$. $\square$