Access-Control Policies via Belnap Logic: Effective and Efficient Composition and Analysis

Glenn Bruns
Bell Labs, Alcatel-Lucent
grb@bell-labs.com

Michael Huth
Dept. of Computing, Imperial College London
mrh@doc.ic.ac.uk

Abstract

It is difficult to develop and manage large, multi-author access control policies without a means to compose larger policies from smaller ones. Ideally, an access-control policy language will have a small set of simple policy combinators that allow for all desired policy compositions. In [5], a policy language was presented having policy combinators based on Belnap logic, a four-valued logic in which truth values correspond to policy results of "grant", "deny", "conflict", and "undefined". We show here how policies in this language can be analyzed, and study the expressiveness of the language. To support policy analysis, we define a query language in which policy analysis questions can be phrased. Queries can be translated into a fragment of first-order logic for which satisfiability and validity checks are computable by SAT solvers or BDDs. We show how policy analysis can then be carried out through model checking, validity checking, and assume-guarantee reasoning over such translated queries. We also present static analysis methods for the particular questions of whether policies contain gaps or conflicts. Finally, we establish expressiveness results showing that all data independent *policies can be expressed in our policy language.*

1 Introduction

The problem we address here is composition in access control policy languages. We take it as a given that policy languages must support policy creation through composition of sub-policies, e.g., as occurring in hierarchical organizations or in secure composition of web services. A policy language should ideally provide a small set of policy composition operators that are simple to understand and use, expressive enough to capture all needed policy compositions, flexible enough to compose the basic rules within a policy as well as whole policies, and based on a mathematical foundation that provides for intellectual tractability and efficient analysis.

Let us consider the semantic objects that expressions in a policy language will denote. Ultimately, a policy is a predicate on access requests. An access-control system receives a request and must make a decision. Think of the justification behind such a decision. If there is reason, or evidence, for granting a request, and no evidence for denying, then the request should be granted. Conversely, if there is only evidence for denying, then the request should be denied. Since a system does not have the luxury to be able to respond to a request with "I don't know" or "I'm confused", it would seem that in the face of no evidence, or conflicting evidence, that the system should deny.

However, in composing policies it makes sense to take a broader view, in which a policy can produce more than just "grant" or "deny". Consider two policies, A and B. Suppose an access request is received for which A produces "grant" and B produces "deny". The only safe response for the combined policy seems to be "deny". But suppose B returns "deny" because there is no evidence for granting or denying, while A returns "grant" because there *is* evidence for granting. Then "grant" seems the correct response for the combined policy. The combined policy could produce this result if policy B did not have to produce a default response of "deny" in the face of no evidence, and could instead produce "undefined". In [15], Halpern and Weissman make this point in the context of a library policy example.

An additional policy response of "conflict" can also help in policy composition. Suppose again that policy A returns "grant" and B returns "deny". We may decide to return "deny" for the result of the combined policy. However, resolving the conflict in this way can mask what may be an error by the policy writer or in the policy requirements. Furthermore, by building a conflict resolution strategy into every policy composition mechanism, we lose the ability to apply a single conflict resolution strategy over a large policy composed of many parts. If several policy composition mechanisms are to be used, each can be simpler if not required to do the job of conflict resolution. Finally, the "conflict" response arises naturally when combining the "information content" of policy responses. Allowing "con-

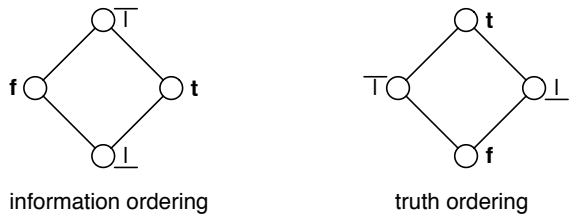

Figure 1. The information and truth orderings on elements of the Belnap space.

flict" as a policy response leads to simplicity and elegance in the formal treatment of access-control policies.

If policies can provide four results, how should the results be combined in policy composition? A natural idea is to think of these four values as truth values, with "grant" as truth and "deny" as falsity, and to combine them with operators of four-valued propositional logic. We write $\mathbf{t}$ for value "grant", $\mathbf{f}$ for "deny", $\bot$ for "undefined", and $\top$ for "conflict". We define a truth ordering $\leq_t$ on these values, with $\mathbf{t}$ the greatest value, $\mathbf{f}$ the least value, and $\top$ and $\bot$ mutually incomparable but less than $\mathbf{t}$ and greater than $\mathbf{f}$ (see Fig. 1). We take the conjunction (written $\wedge$) of two values to be their greatest lower bound, and the disjunction (written $\vee$) to be their least upper bound. For example, the conjunction $\mathbf{t} \wedge \top$ is $\top$, and the disjunction $\mathbf{f} \vee \bot$ is $\bot$. We can apply these operations to policy results. So if policy A grants a request (produces $\mathbf{t}$), and policy B is in conflict on the request (produces $\top$), then policy $A \wedge B$ produces $\top$. Policy negation (written $\neg$) can be defined similarly: it swaps $\mathbf{t}$ and $\mathbf{f}$, and leaves $\bot$ and $\top$ unchanged. However, some useful policy operations cannot be captured using these logical operations. For example, a natural composition operation will sum the "information content" of two policy responses. If policy A produces $\bot$, and B produces $\mathbf{t}$, we may want the combined policy to produce $\mathbf{t}$. Therefore we define an information ordering $\leq_k$ (index k stands for "knowledge") over the four policy result values, with $\top$ the greatest and $\bot$ the least value (see Fig. 1). Relative to this ordering we can again define an operation (written $\oplus$) that gives the least upper bound and another (written $\otimes$) that gives the greatest lower bound. The least upper bound $\mathbf{t} \oplus \bot$ is $\mathbf{t}$, and $\mathbf{t} \oplus \mathbf{f}$ is $\top$.

A logic based on these four values and the operators we have mentioned was developed by Belnap [2]. In [5] we presented a policy language based on Belnap logic, which we here refer to as PBel (for "Policy language based on Belnap logic", and pronounced "pebble"). Composition in PBel has many virtues. It is denotational, as the meaning of a policy comes from the meaning of its sub-policies. The policy combinators have clear and simple meanings, and apply equally well to the composition of large policies and the composition of single rules within a policy. From the operators in Belnap logic we can derive a wide range of policy operators, including those used in popular access-control policy languages like XACML [9] and Cisco's IOS firewall policy language [24]. Furthermore, policy composition in PBel is based on established mathematical concepts and provides for the efficient analysis of conflict-freedom and other important properties of policies.

In this paper the focus is policy analysis. We develop a framework supporting a range of policy analysis questions and verification methods. We define a propositional policy query language in which policy analysis questions can be formalized. The atomic propositions of this language ask whether two policies are related by the truth or information ordering. A policy query is then translated to a first-order logic constraint over the access predicate symbols of the policies appearing in the query. With these constraints we can answer policy analysis questions through model checking, validity checking, or assume-guarantee reasoning. For example, suppose we want to check that a policy A has no conflicts on a policy interpretation $\mathcal{M}$ (which interprets predicates on access requests that may appear in a policy expression). We express the property of conflict-freedom as a query ϕ, from ϕ obtain a first-order constraint c by translation, and then model check to see whether $\mathcal{M}$ satisfies c. The model check succeeds iff policy A is conflict-free. For validity checking and assume-guarantee reasoning we use SAT solving or BDDs on propositional formulas derived from the first-order constraints.

For the important policy properties of conflict-freedom and gap-freedom, we additionally provide static analyses through type systems. Finally, we consider the question of the expressiveness of PBel. We show that PBel expresses exactly *data-independent* functions, functions that map access requests to the same value if these requests cannot be distinguished by access predicates of the ambient system. We also sketch an extension of PBel that supports a composition called "closure" by Woo and Lam [25]. This allows one policy's reply to a request to depend on another policy's reply to a different request.

Outline. The paper is organized as follows. In Section 2 we present PBel. In Section 3 we define a query language in which policy analysis questions can be phrased. In Section 4 we show how a policy query can be translated to a logical constraint on access predicates of the policies appearing in the query. In Section 5 we present static analyses for gap and conflict-freedom of policies. In Section 6 we show how our machinery can be applied to the analysis of firewall policies. In Section 7 we present a method for assume-guarantee reasoning about policies, allowing for more precise property verification. In Section 8 we study the expressiveness of PBel. In Section 9 we discuss related work. In

p, q ::=	*Policy*
b if ap_i	Basic policy
$\neg p$	Logical negation
$p \wedge q$	Logical meet
$p \supset q$	Implication
$p \oplus q$	Nondeterministic choice
$p[v \mapsto q]$	Refinement

Figure 2. Policy Language PBel: the ap_i are access predicates, $b \in \{\mathbf{f}, \mathbf{t}\}$, and $v \in \{\bot, \top\}$.

Section 10 we sketch language extensions and point out future work. Finally, in Section 11 we conclude.

2 Policies

An access control policy defines the response an access control system should make to an access request (we usually refer to an access request simply as an "access"). We take responses of an access control system to be the values $\mathbf{t}$, $\mathbf{f}$, $\top$, and $\bot$ of the aforementioned *Belnap space*, which we write as $\mathbf{Four}$. These values and their two orderings form a distributive, interlaced bilattice [14, 13]. Thus Belnap logic has many convenient properties, for example: $\wedge$ and $\vee$ distribute as in propositional logic, as do $\otimes$ and $\oplus$, and all these operators are also monotone to both the $\leq_k$ and $\leq_t$ orderings.

Operator $\supset$ is an extension of implication to the Belnap space [1]. Expression $a \supset b$ with $a, b \in \mathbf{Four}$ yields b if a is greater or equal to $\mathbf{t}$ in the information ordering, and yields $\mathbf{t}$ otherwise. We additionally define an "overwriting" operator $[y \mapsto z]$ for elements of $\mathbf{Four}$. Expression $x[y \mapsto z]$ yields x if $x \neq y$, and z otherwise. Figure 2 gives the syntax of PBel, our access control policy language. Informally, a PBel expression is interpreted as a mapping from accesses to elements of $\mathbf{Four}$. If policy p produces result x on access a and policy q produces result y on a, then $p \wedge q$ produces result $x \wedge y$ on a. The other Belnap operators are similarly interpreted on policies as the pointwise extensions of their logical meaning. The intuition behind these policy operators is that:

- $\neg p$ denies an access iff p grants it (and vice versa)
- $p \wedge q$ grants an access iff both p and q grant it, and denies an access iff at least one of p and q denies it
- $p \supset q$ grants an access iff p does not grant it or q does grant it, and $p \supset q$ denies an access if p grants it and q denies it
- $p \oplus q$ grants (resp. denies) an access iff p or q grants (resp. denies) it
- $p[\bot \mapsto q]$ grants an access iff either p grants it, or p has no information on that access and q grants it; $p[\bot \mapsto q]$ denies an access iff p denies it, or p has no information on that access and q denies it
- $p[\top \mapsto q]$ grants an access iff either p grants it and does not deny it at the same time, or q grants it; $p[\top \mapsto q]$ denies an access iff either p denies it and does not grant it at the same time, or q denies it

Arieli and Avron showed [1] that the set $\{\neg, \oplus, \supset, \bot\}$ of Belnap operators is functionally complete for $\mathbf{Four}$: all functions of type $\mathbf{Four}^k \to \mathbf{Four}$ can be expressed as expressions in this fragment of Belnap logic. Therefore in examples we will use $\otimes$ and other derivable operators freely. Indeed, operator $[v \mapsto q]$ is itself derivable from other operators of PBel, but it is so useful that we have included it explicitly in our core policy language.

The only operator in PBel not derived from Belnap logic is the basic PBel expression b if ap_i, where b ranges over $\{\mathbf{f}, \mathbf{t}\}$, and ap_i is an *access predicate*. Informally, the b if ap_i "rule" gives result b for an access satisfying ap_i and gives result $\bot$ otherwise. The idea is that ap_i defines the domain of accesses covered by the rule, so that $\bot$ is produced on accesses outside of the rule's domain.

For generality we take accesses to be atomic entities, abstracting from the exact form of accesses, which will vary between applications. Often accesses are modeled as tuples of the form $\langle \mathsf{subject}, \mathsf{action}, \mathsf{object} \rangle$, for example $\langle \mathsf{librarian}, \mathsf{write}, \mathsf{catalog} \rangle$, but in general an access could contain a time value, a history of access requests, values representing object states, or any other values needed to make access control decisions. By using access predicates we are not tied to a particular structure for accesses, or to a particular logic for forming predicates on accesses. We expect an instantiation of our language for some application domain to include a sub-language in which to express access predicates. A specialized language might, e.g., treat accesses as triples (as above, or with an additional state tag), allow the definition of user, subject, and action domains, and provide a logical language for access predicates.

If ap_1 is true of accesses in which a librarian writes to the card catalog, and ap_2 is true of accesses in which a library user writes to the card catalog, then a simple library policy might be $(\mathbf{t}$ if $ap_1) \oplus (\mathbf{f}$ if $ap_2)$. This policy grants access to librarians seeking to write to the card catalog, denies access to users seeking to write to the card catalog, and produces result $\top$ to a librarian who is also a library user seeking to write to the card catalog. But policy $(\mathbf{t}$ if $ap_1) \wedge (\mathbf{f}$ if $ap_2)$ would deny catalog writes by the librarian/user.

To give formal meaning to PBel expressions, we define a class of accesses and a set of access predicates over this class. For convenience we work with a countable set $\{ap_i \mid i \geq 1\}$ of symbols for access predicates. Examples may use

$$
\begin{array}{rcl}
[\![b \text{ if } ap_i]\!]_{\mathcal{M}}(a) & \stackrel{\text{def}}{=} & \begin{cases} b & \text{if } a \in ap_i^{\mathcal{M}} \\ \bot & \text{otherwise} \end{cases} \\
[\![\neg p]\!]_{\mathcal{M}}(a) & \stackrel{\text{def}}{=} & \neg [\![p]\!]_{\mathcal{M}}(a) \\
[\![p \wedge q]\!]_{\mathcal{M}}(a) & \stackrel{\text{def}}{=} & [\![p]\!]_{\mathcal{M}}(a) \wedge [\![q]\!]_{\mathcal{M}}(a) \\
[\![p \supset q]\!]_{\mathcal{M}}(a) & \stackrel{\text{def}}{=} & [\![p]\!]_{\mathcal{M}}(a) \supset [\![q]\!]_{\mathcal{M}}(a) \\
[\![p \oplus q]\!]_{\mathcal{M}}(a) & \stackrel{\text{def}}{=} & [\![p]\!]_{\mathcal{M}}(a) \oplus [\![q]\!]_{\mathcal{M}}(a) \\
[\![p[v \mapsto q]]\!]_{\mathcal{M}}(a) & \stackrel{\text{def}}{=} & [\![p]\!]_{\mathcal{M}}(a)[v \mapsto [\![q]\!]_{\mathcal{M}}(a)]
\end{array}
$$

Figure 3. Meaning $[\![p]\!]_{\mathcal{M}}(a)$ of PBel expression p at access a of access-control state $\mathcal{M}$.

meaningful names for some of these access predicates.

Definition 1 *An* access-control state $\mathcal{M} = (A_{\mathcal{M}}, \{ap_i^{\mathcal{M}} \mid i \geq 1\})$ *is a non-empty set $A_{\mathcal{M}}$ of accesses with a predicate $ap_i^{\mathcal{M}} \subseteq A_{\mathcal{M}}$ for each $i \geq 1$.*

An access-control state can serve as a model of first-order logic, in which the accesses are the objects, and $ap_i^{\mathcal{M}}$ is the interpretation of unary predicate symbol ap_i. We call it an "access-control state" since one might want to allow policies in which operations can modify access predicates or create or delete accesses. We assume that every access-control state $\mathcal{M}$ includes an access predicate, which we write as true, that holds of all accesses in $A_{\mathcal{M}}$.

We interpret a PBel expression p relative to an access-control state $\mathcal{M}$ as a function $[\![p]\!]_{\mathcal{M}} \colon A_{\mathcal{M}} \to \mathbf{Four}$ from accesses to elements of $\mathbf{Four}$, as defined in Fig. 3. The operators $\neg$, $\wedge$, $\supset$, $\oplus$, and $x[y \mapsto z]$ on the right-hand side of equations in Fig. 3 are operators on the Belnap space discussed earlier. We often write $p(a)$ as shorthand for $[\![p]\!]_{\mathcal{M}}(a)$ when intended access-control state $\mathcal{M}$ is clear.

Example 1 *We show how some common ideas of policy composition can be expressed in PBel.*

- *"Priority composition" of p and q gives p's result if it is not $\bot$, and otherwise gives q's result. This idea can be written $p[\bot \mapsto q]$, which we abbreviate as $p > q$.*
- *A "disjoint policy" composition gives $\top$ if both p and q produce a non-$\bot$ result, gives $\bot$ if both policies produce $\bot$, and otherwise gives the result of the policy that does not produce $\bot$. This idea can be written $(p \oplus q) \oplus ((p \oplus \neg p) \otimes (q \oplus \neg q))$. The expression $p \oplus \neg p$ yields $\bot$ if p yields $\bot$, and otherwise yields $\top$.*
- *A "constant" policy that grants all accesses can be written* ($\mathbf{t}$ if true). *We abbreviate* ($\mathbf{t}$ if true) *as* $\mathbf{t}$*, for example in expression $p[\top \mapsto \mathbf{t}]$. Similarly* $\mathbf{f}$ *is shorthand for* ($\mathbf{f}$ if true).

ϕ, ψ ::=	*Query*
$p \leq_t q$	Truth ordering
$p \leq_k q$	Information ordering
$\neg(p \leq_t q)$	Negation Truth ordering
$\neg(p \leq_k q)$	Negation Information ordering
$\phi \wedge \psi$	Conjunction
$\phi \vee \psi$	Disjunction

Figure 4. Query Language: p and q range over PBel expressions.

Because the Belnap space is a distributive, interlaced bilattice, and because policies are defined as the functions from accesses to the Belnap space, we have by a result of Fitting's (see [13]) that policies themselves form a distributive, interlaced bilattice, where $p \leq_k q$ if $p(a) \leq_k q(a)$ for every access a, and similarly for the $\leq_t$ ordering on policies. Thus policies inherit all the useful properties from the Belnap space, such as that $\vee$ and $\wedge$ distribute.

3 Queries

Properties about policies and their relationships can be expressed as *queries*, propositional formulas in which the atomic propositions concern the pointwise truth or information ordering among policies [5]. Figure 4 gives the abstract syntax for queries (in negation normal form for technical convenience). We overload the symbols for negation ($\neg$) and conjunction ($\wedge$) as context will show how they apply. A query is interpreted relative to an access-control state.

Definition 2 *Let ϕ be a query. An access-control state $\mathcal{M}$ satisfies ϕ, written $\mathcal{M} \models \phi$, according to the following structural induction, where $* \in \{k, t\}$:*

- $\mathcal{M} \models p \leq_* q$ *Iff for all* $a \in A_{\mathcal{M}}$ *we have* $[\![p]\!]_{\mathcal{M}}(a) \leq_* [\![q]\!]_{\mathcal{M}}(a)$
- $\mathcal{M} \models \neg(p \leq_* q)$ *Iff there is some* $a \in A_{\mathcal{M}}$ *with* $[\![p]\!]_{\mathcal{M}}(a) \not\leq_* [\![q]\!]_{\mathcal{M}}(a)$
- $\mathcal{M} \models \phi \wedge \psi$ *Iff* $\mathcal{M} \models \phi$ *and* $\mathcal{M} \models \psi$
- $\mathcal{M} \models \phi \vee \psi$ *Iff* $\mathcal{M} \models \phi$ *or* $\mathcal{M} \models \psi$

We now present queries for some examples of policy specification and analysis.

Gap and conflict analysis. Conflict (resp. gap) analysis asks whether a policy can ever return a value $\top$ or $\bot$ (resp.).

We claim that the query $p \leq_k p[\top \mapsto \mathbf{f}]$ correctly specifies conflict-freedom of policy p, and prove the correctness of this claim. A similar query is also proved to correctly characterize gap-freedom.

Definition 3 *For a PBel expression p and access-control state $\mathcal{M}$, p is* gap-free (*resp.,* conflict-free) in $\mathcal{M}$ *iff for no access a in $A_{\mathcal{M}}$ is $[\![p]\!]_{\mathcal{M}}(a) = \bot$ (resp., $[\![p]\!]_{\mathcal{M}}(a) = \top$).*

Theorem 1 *For every access-control state $\mathcal{M}$ and PBel expression p we have:*

1. *p is gap-free in $\mathcal{M}$ iff $\mathcal{M} \models p \leq_t p[\bot \mapsto \mathbf{f}]$*
2. *p is conflict-free in $\mathcal{M}$ iff $\mathcal{M} \models p \leq_k p[\top \mapsto \mathbf{f}]$.*

Change management. In [12], Fisler *et al.* consider the problem of change-impact analysis of access-control policies. When a policy p is updated, one may want to confirm that too many permissions were not added, either unintentionally or maliciously. For example, we may require that the legal scope of any new permissions is bounded by an access predicate ap_1. So the query $p \leq_t (p \oplus (\mathbf{t} \text{ if } ap_1))$ should hold in all access-control states. Or suppose we want to refine policy $(p_1 > p_2)$ into $(p_1 > (p_2 \wedge q))$ such that the new policy is gap-free. Can one synthesize a PBel expression q that suits these needs? This problem of *policy synthesis* is the subject of future work.

4 From Queries to Constraints

The truth of a query could be evaluated by a "model check" of the query, through the first-order logic semantics for $\models$. However, such model checking offers limited value for the analysis of policies and their compositions. There may be a very large or infinite access-control state space, so a literal interpretation of quantifiers in $\mathcal{M} \models \phi$ may be impossible or too costly to compute. In addition, access-control states may change with any decision to grant an access, as that may grant the privilege to change access rights. Therefore queries may have to be checked for an entire region of access-control states (e.g. as for discretionary access control models [20]), and so model checking may have to be replaced with satisfiability or validity checking.

The alternative approach proposed here for evaluating a query is to translate the query into a constraint on the basic access predicates appearing in the policies of the query. For example, an analysis of this query

$$\mathbf{f} \text{ if } ap_1 \leq_k \neg[(\mathbf{t} \text{ if } ap_2) \wedge (\mathbf{f} \text{ if } ap_3)] \qquad (1)$$

should reveal that it holds in an access-control state $\mathcal{M}$ iff access predicate $ap_1^{\mathcal{M}}$ is empty.

c, c' ::=	*Constraint*
$\mathtt{tt}$	Truth
$ap_i(x)$	Access predicate as atom
$c \wedge c'$	Conjunction
$c \vee c'$	Disjunction
$\neg c$	Negation
$\forall x.\, c$	Universal quantification
$\exists x.\, c$	Existential quantification

Figure 5. Constraint Language: ap_i and x denote access predicates and variables (resp.).

More precisely, from each query ϕ is computed a first-order logical formula in which atomic expressions are access predicates $ap_i(x)$ and which is *nesting-free* (no quantifiers are within scope of other quantifiers). Figure 5 shows the abstract syntax of constraints. We freely use additional logical operators, e.g. implication ($\rightarrow$) and falsity ($\mathtt{ff}$), that can be derived from the operators of the language in the usual manner. Our analysis translates all queries into closed, nesting-free formulas with x as sole variable; we call such formulas *nesting-free constraints*.

The meaning of constraint c is given by the standard semantics of first-order logic, using a first-order valuation v over $\mathcal{M}$ that maps all variable symbols occurring freely in c to elements of $\mathcal{M}$, and may thus be a partial function. We write $v[x := a]$ for the valuation that is like v except that x maps to access a, and write $()$ for the empty valuation.

Definition 4 *Let $\mathcal{M}$ be an access-control state, c a constraint, and v be a first-order valuation over $\mathcal{M}$ defined for all x occurring in c. Then the satisfaction $\mathcal{M}, v \models c$ is defined by structural induction, with truth constants and propositional combinators taking their usual meaning, and*

- $\mathcal{M}, v \models ap_i(x)$ *Iff* $v(x) \in ap_i^{\mathcal{M}}$
- $\mathcal{M}, v \models \forall x.\, c$ *Iff for all* a: $\mathcal{M}, v[x := a] \models c$
- $\mathcal{M}, v \models \exists x.\, c$ *Iff for some* a: $\mathcal{M}, v[x := a] \models c$

For a closed constraint c the valuation v is irrelevant, so we can write $\mathcal{M} \models c$ instead of $\mathcal{M}, v \models c$. Subsequently we also use $\mathcal{M}, a \models c$ as shorthand for $\mathcal{M}, ()[x := a] \models c$ if c is a constraint with at most a single free variable x.

The translation from queries to constraints that we shall soon present uses an auxiliary function, defined in Fig. 6, that translates PBel expressions p to quantifier-free constraints with x as sole variable. (Throughout $\Uparrow$ has higher binding priority than $\vee$, $\wedge$, and $\rightarrow$, but $\neg$ binds more tightly than $\Uparrow$). This translation is static and independent of access-control states. But it gives, for every policy p and b in $\{\mathbf{f}, \mathbf{t}\}$, a constraint $p \Uparrow b$ that holds of access a just if $p(a)$ is at least b in the information ordering.

$$
\begin{array}{rcl}
(b' \text{ if } ap_i) \Uparrow b & = & \begin{cases} ap_i(x) & \text{if } b = b' \\ \mathsf{ff} & \text{otherwise} \end{cases} \\
(\neg p) \Uparrow b & = & p \Uparrow \neg b \\
(p \wedge q) \Uparrow \mathbf{f} & = & p \Uparrow \mathbf{f} \vee q \Uparrow \mathbf{f} \\
(p \wedge q) \Uparrow \mathbf{t} & = & p \Uparrow \mathbf{t} \wedge q \Uparrow \mathbf{t} \\
(p \supset q) \Uparrow \mathbf{f} & = & p \Uparrow \mathbf{t} \wedge q \Uparrow \mathbf{f} \\
(p \supset q) \Uparrow \mathbf{t} & = & \neg(p \Uparrow \mathbf{t}) \vee q \Uparrow \mathbf{t} \\
(p \oplus q) \Uparrow b & = & p \Uparrow b \vee q \Uparrow b \\
p[\top \mapsto q] \Uparrow b & = & p \Uparrow b \wedge [\neg(p \Uparrow \neg b) \vee q \Uparrow b] \\
p[\bot \mapsto q] \Uparrow b & = & p \Uparrow b \vee [\neg(p \Uparrow \neg b) \wedge q \Uparrow b]
\end{array}
$$

Figure 6. Constraint $p \Uparrow b$ for PBel expression p and $b \in \{\mathbf{f}, \mathbf{t}\}$. Access a satisfies $p \Uparrow b$ iff $p(a) \geq_k b$.

$$
\begin{array}{rcl}
\| \, p \leq_t q \, \| & = & \forall x. \, ([q \Uparrow \mathbf{f} \rightarrow p \Uparrow \mathbf{f}] \wedge [p \Uparrow \mathbf{t} \rightarrow q \Uparrow \mathbf{t}]) \\
\| \, p \leq_k q \, \| & = & \forall x. \, ([p \Uparrow \mathbf{f} \rightarrow q \Uparrow \mathbf{f}] \wedge [p \Uparrow \mathbf{t} \rightarrow q \Uparrow \mathbf{t}]) \\
\| \, \neg(p \leq_t q) \, \| & = & \exists x. \, \neg([q \Uparrow \mathbf{f} \rightarrow p \Uparrow \mathbf{f}] \wedge [p \Uparrow \mathbf{t} \rightarrow q \Uparrow \mathbf{t}]) \\
\| \, \neg(p \leq_k q) \, \| & = & \exists x. \, \neg([p \Uparrow \mathbf{f} \rightarrow q \Uparrow \mathbf{f}] \wedge [p \Uparrow \mathbf{t} \rightarrow q \Uparrow \mathbf{t}]) \\
\| \, \psi \wedge \psi \, \| & = & \| \, \phi \, \| \wedge \| \, \psi \, \| \\
\| \, \psi \vee \psi \, \| & = & \| \, \phi \, \| \vee \| \, \psi \, \|
\end{array}
$$

Figure 7. Constraint $\| \, \phi \, \|$ for query ϕ.

Theorem 2 *Suppose p is a PBel expression, $b \in \{\mathbf{f}, \mathbf{t}\}$, $\mathcal{M}$ is an access-control state, and $a \in A_{\mathcal{M}}$. Then $p \Uparrow b$ is a quantifier-free constraint with x as sole variable such that*

$$\mathcal{M}, a \models p \Uparrow b \qquad \textit{Iff} \qquad [\![p]\!]_{\mathcal{M}}(a) \geq_k b$$

Figure 7 shows the top-level translation from a query ϕ to a nesting-free constraint $\| \, \phi \, \|$ in a *negation normal form for quantifiers*: no quantifier is within the scope of a negation (but propositional operators may be in such a scope). Positive and negative literals $p \leq_* q$ and $\neg(p \leq_* q)$ are compiled using the predicates $p \Uparrow b$ and the following characterization of the truth and information orderings on $\mathbf{Four}$:

Proposition 1 *For all x and y of* $\mathbf{Four}$ *we have*

1. $x \leq_t y$ *Iff* $(y \geq_k \mathbf{f} \Rightarrow x \geq_k \mathbf{f}) \,\&\, (x \geq_k \mathbf{t} \Rightarrow y \geq_k \mathbf{t})$
2. $x \leq_k y$ *Iff* $(x \geq_k \mathbf{f} \Rightarrow y \geq_k \mathbf{f}) \,\&\, (x \geq_k \mathbf{t} \Rightarrow y \geq_k \mathbf{t})$

This translation of queries to constraints satisfies the desired correctness condition: query ϕ holds at an access-control state iff the constraint obtained from ϕ does.

Theorem 3 *Let $\mathcal{M}$ be an access-control state and ϕ be a query. Then $\| \, \phi \, \|$ is a nesting-free constraint in negation normal form such that*

$$\mathcal{M} \models \| \, \phi \, \| \;\; \textit{Iff} \;\; \mathcal{M} \models \phi$$

Example 2 *We compute constraint $\| \, \phi \, \|$ for the query of form $p \leq_k q$ in (1) above, replacing constraints with equivalent ones if these replacements are easily detected by SAT solvers or theorem provers (e.g. changing $\mathsf{ff} \wedge \psi$ to ff):*

- $p \Uparrow \mathbf{t} = \mathsf{ff}$
- $p \Uparrow \mathbf{f} = ap_1(x)$
- $q \Uparrow \mathbf{t} = [(\mathbf{t} \text{ if } ap_2) \wedge (\mathbf{f} \text{ if } ap_3)] \Uparrow \mathbf{f} = (\mathbf{t} \text{ if } ap_2) \Uparrow \mathbf{f} \vee (\mathbf{f} \text{ if } ap_3) \Uparrow \mathbf{f} = \mathsf{ff} \vee ap_3(x) = ap_3(x)$
- $q \Uparrow \mathbf{f} = [(\mathbf{t} \text{ if } ap_2) \wedge (\mathbf{f} \text{ if } ap_3)] \Uparrow \mathbf{t} = (\mathbf{t} \text{ if } ap_2) \Uparrow \mathbf{t} \wedge (\mathbf{f} \text{ if } ap_3) \Uparrow \mathbf{t} = ap_2(x) \wedge \mathsf{ff} = \mathsf{ff}$
- $\| \, \phi \, \| = \forall x. \, ([ap_1(x) \rightarrow \mathsf{ff}] \wedge [\mathsf{ff} \rightarrow ap_3(x)])$*, which simplifies to* $\forall x. \neg ap_1(x)$.

By Theorem 3 we therefore know that $\mathcal{M} \models p \leq_k q$ holds iff $\mathcal{M} \models \forall x. \neg ap_1(x)$ holds, i.e. iff $ap_1^{\mathcal{M}} = \emptyset$. In particular, this is independent of the meaning of ap_2 and ap_3.

4.1 Symbolic optimizations

Consider the query $p \leq_t q$, where p and q are understood as parameters for PBel expressions. We can apply the translation rules in Fig. 7 to the query to obtain a nesting-free constraint $\| \, p \leq_t q \, \|$ with x as sole variable for the first-order logic quantifiers but with $p \Uparrow b$ and $q \Uparrow b$ as formal parameters. Any simplifications of this symbolic version of $\| \, p \leq_t q \, \|$ will therefore be optimizations for policy analyses of $\| \, p \leq_t q \, \|$ when p and q are instantiated with actual PBel expressions. This optimization will also apply to the assume-guarantee reasoning of Section 7.

Example 3 *We compute the constraint $\| \, p \leq_k p[\top \mapsto \mathbf{f}] \, \|$ for conflict analysis symbolically in terms of $p \Uparrow \mathbf{f}$ and $p \Uparrow \mathbf{t}$. We have $p[\top \mapsto \mathbf{f}] \Uparrow \mathbf{t} = p \Uparrow \mathbf{t} \wedge (\neg(p \Uparrow \mathbf{f}) \vee (\mathbf{f} \text{ if } \mathsf{true}) \Uparrow \mathbf{t})$, which simplifies to $p \Uparrow \mathbf{t} \wedge \neg(p \Uparrow \mathbf{f})$, since $(\mathbf{f} \text{ if } \mathsf{true}) \Uparrow \mathbf{t}$ simplifies to ff. Also, $p[\top \mapsto \mathbf{f}] \Uparrow \mathbf{f} = p \Uparrow \mathbf{f} \wedge (\neg(p \Uparrow \mathbf{t}) \vee (\mathbf{f} \text{ if } \mathsf{true}) \Uparrow \mathbf{f})$, which simplifies to $p \Uparrow \mathbf{f}$ since $(\mathbf{f} \text{ if } \mathsf{true}) \Uparrow \mathbf{f}$ simplifies to tt. Then we get $\| \, p \leq_k p[\top \mapsto \mathbf{f}] \, \| = \forall x. [p \Uparrow \mathbf{f} \rightarrow p \Uparrow \mathbf{f}] \wedge [p \Uparrow \mathbf{t} \rightarrow (p \Uparrow \mathbf{t} \wedge \neg(p \Uparrow \mathbf{f}))]$, which simplifies to $\forall x. [p \Uparrow \mathbf{t} \rightarrow \neg(p \Uparrow \mathbf{f})]$. This confirms the intuition behind conflict analysis: a policy p is conflict-free whenever $p(a) \geq_k \mathbf{t}$ implies $p(a) \not\geq_k \mathbf{f}$.*

Let p be $p_1 \oplus p_2$. Then $\| \, p \leq_k p[\top \mapsto \mathbf{f}] \, \|$ simplifies to $\forall x. [\neg(p_1 \Uparrow \mathbf{t} \vee p_2 \Uparrow \mathbf{t}) \vee \neg(p_1 \Uparrow \mathbf{f} \vee p_2 \Uparrow \mathbf{f})]$. So $p_1 \oplus p_2$ contains conflict just if there exists an access for which at least one of p_1 and p_2 grants and at least one denies.

Example 4 *A symbolic optimization for gap analysis unfolds* $\| p \leq_t p[\bot \mapsto \mathbf{f}] \|$ *symbolically to* $\forall x.[p \Uparrow \mathbf{t} \vee p \Uparrow \mathbf{f}]$ *(done as in Example 3). So a policy* p *has a gap if it satisfies* $\exists x.\, [\neg(p \Uparrow \mathbf{t} \wedge \neg(p \Uparrow \mathbf{f})]$. *Suppose we instantiate* p *with* $(\mathbf{f}\ \mathsf{if}\ ap_1) > (\mathbf{f}\ \mathsf{if}\ ap_2)$. *Then* $\neg(p \Uparrow \mathbf{t}) \wedge \neg(p \Uparrow \mathbf{f})$ *simplifies to* $\neg ap_1(x) \wedge \neg ap_2(x)$. *So policy* p *has a gap in access-control states* $\mathcal{M}$ *for which* $\mathcal{M} \models \exists x.\, [\neg ap_1(x) \wedge \neg ap_2(x)]$.

Example 5 *Finally, consider the positive query* $\phi = (p \leq_k q) \wedge (p \leq_t q)$*, which states that policy* q *is more defined and more permissive than policy* p. *The formula* $\| \phi \|$ *simplifies to* $\forall x.[(p \Uparrow \mathbf{t} \rightarrow q \Uparrow \mathbf{t}) \wedge (p \Uparrow \mathbf{f} \leftrightarrow q \Uparrow \mathbf{f})]$.

Figure 8 summarizes some facts about gap and conflict-freedom obtained by symbolic optimization and appeal to Theorems 1 and 3. The constraint for conflict-freedom of $p_1 > \cdots > p_n$ in Fig. 8 says that $p_1 > \cdots > p_n$ is conflict-free iff every sub-policy p_i is either conflict-free itself or has some predecessor that does not return $\bot$. In the special case where each p_i is itself conflict-free, then clearly so is $p_1 > \cdots > p_n$. We summarize:

Proposition 2 *In Figure 8, a PBel expression in the first column is gap-free in an access-control state* $\mathcal{M}$ *iff* $\mathcal{M}$ *satisfies the constraint given in the second column, and is conflict-free in* $\mathcal{M}$ *iff* $\mathcal{M}$ *satisfies the constraint given in the third column.*

5 Static Analysis

Our policy language and its semantics have compositional gap and conflict analyses based on type inference in simple type systems. Figure 9 shows such a sound type system for deriving judgments $\mathsf{CF}(p)$ that ensure conflict-freedom of policy p. Derivation trees for $\mathsf{CF}(p)$ in that type system are unique, i.e. all type inference rules are invertible.

Theorem 4 *Let* p *be a PBel expression such that* $\mathsf{CF}(p)$ *can be derived from the type inference rules in Fig. 9. Then* p *is conflict-free at all access-control states* $\mathcal{M}$. *In particular, all policies with no occurrence of* $\oplus$ *are conflict-free.*

As a consequence, $\oplus$ cannot be expressed by any combination of other policy operators in PBel since $(\mathbf{f}\ \mathsf{if}\ ap_1) \oplus (\mathbf{t}\ \mathsf{if}\ ap_1)$ is not conflict-free.

The type system for $\mathsf{CF}(p)$ is sound but incomplete: $(\mathbf{f}\ \mathsf{if}\ ap_1) \oplus (\mathbf{f}\ \mathsf{if}\ ap_2)$ is obviously conflict-free, but our type system has no inference rule for $\oplus$ nor can there be a sound compositional rule for conflict analysis of $\oplus$. Fortunately, Theorem 4 remains valid as long as assumptions in a derivation tree can be assured to be conflict-free *by other means*. Policy $p = (\mathbf{f}\ \mathsf{if}\ ap_1) \oplus (\mathbf{f}\ \mathsf{if}\ ap_2)$, e.g., is conflict-free as $\| p \leq_k p[\top \mapsto \mathbf{f}] \|$ is valid. So $p \wedge (\mathbf{f}\ \mathsf{if}\ ap_3)$ is conflict-free by the inference rule for $\wedge$ in Fig. 9.

Figure 10 shows a type inference system for judgments $\mathsf{GF}(p)$, which assert policy p is gap-free. Its sole axiom concerns constant policies that either deny or grant all accesses. But its inference rules are sound if their premises are gap-free, so assumptions $\mathsf{GF}(p)$ may be used for inferences whenever p is shown to be gap-free through other means.

Theorem 5 *Let* $\mathsf{GF}(p)$ *be derived from the type inference rules in Fig. 10 through assumptions* $\mathsf{GF}(p_i)$ *with* $i = 1, \ldots, n$. *If constraint* $\| p_i \leq_t p_i[\bot \mapsto \mathbf{f}] \|$ *is valid for* $i = 1, \ldots, n$ *then* p *is gap-free in all access-control states* $\mathcal{M}$.

Example 6 *Let* p *be any PBel expression and* $q = (\mathbf{t}\ \mathsf{if}\ ap_1) \supset (\mathbf{t}\ \mathsf{if}\ ap_1)$. *Then* $\| q \leq_t q[\bot \mapsto \mathbf{f}] \|$ *is valid and so* q *is gap-free by Theorem 1. Thus* $p[\top \mapsto q]$ *is gap-free by Theorem 5.*

6 Example: Firewall analysis

We now consider the analysis of firewall policies, which are used to control traffic into or out of a private network. Our analysis uses the results from Fig. 8 to generate optimized constraints whose validity can be shown with a SAT solver. Alternatively, these constraints can be model checked on access-control states or converted into BDDs [6] for the purpose of debugging, or perhaps even implementing, a firewall policy. Our formal model of firewall policies is based on the extended access lists of Cisco's IOS firewall [24], used in Cisco routers and other products. The work of Capretta *et al.* [7] on the analysis of conflict detection in Cisco firewall policies also served as a reference and an inspiration. See the end of this section for a comparison of the work in [7] with the work reported here.

In a firewall policy an access is a packet, and various attributes of the packet are examined in making a policy decision. Typical attributes include host and target addresses and port numbers, as well as service (such as the FTP protocol), so we model an access as a tuple of the form

$$\langle \mathsf{hostIP}, \mathsf{targetIP}, \mathsf{hostPort}, \mathsf{targetPort}, \mathsf{service} \rangle$$

We can think of a firewall policy as a sequence $r_1, \ldots, r_n$ of *rules*, where each rule r_i is a conjunction of basic PBel expressions. Using $b\ \mathsf{if}\ (ap_i \wedge ap_j)$ as shorthand for $(b\ \mathsf{if}\ ap_i) \wedge (b\ \mathsf{if}\ ap_j)$, each rule r_i has the form:

$$r_i = b_i\ \mathsf{if} \bigwedge_{j=1}^{5} ap_{i,j} \tag{2}$$

One could alternatively model a rule as an expression of the form $\bigwedge_{j=1}^{5} (b_{i,j}\ \mathsf{if}\ ap_{i,j})$. Yet, nothing is gained from allowing both a granting and a denying expression within a rule of that form as expressions $(\mathbf{f}\ \mathsf{if}\ ap_1) \wedge (\mathbf{t}\ \mathsf{if}\ ap_2)$ and $\mathbf{f}\ \mathsf{if}\ ap_1$ are easily seen to be semantically equivalent.

policy	constraint for gap-freedom	constraint for conflict-freedom
p	$\forall x.(p \Uparrow \mathbf{t} \vee p \Uparrow \mathbf{f})$	$\forall x.[\neg(p \Uparrow \mathbf{t}) \vee \neg(p \Uparrow \mathbf{f})]$
$p_1 \oplus \cdots \oplus p_n$	$\forall x. \bigvee_{i=1}^{n}(p_i \Uparrow \mathbf{t} \vee p_i \Uparrow \mathbf{f})$	$\forall x.(\neg[\bigvee_{i=1}^{n} p_i \Uparrow \mathbf{t}] \vee \neg[\bigvee_{i=1}^{n} p_i \Uparrow \mathbf{f}])$
$p_1 > \cdots > p_n$	$\forall x. \bigvee_{i=1}^{n}(p_i \Uparrow \mathbf{t} \vee p_i \Uparrow \mathbf{f})$	$\forall x. \bigwedge_{i=1}^{n}[\neg(p_i \Uparrow \mathbf{f}) \vee \neg(p_i \Uparrow \mathbf{t}) \vee \bigvee_{j=1}^{i-1}(p_j \Uparrow \mathbf{f} \vee p_j \Uparrow \mathbf{t})]$
$p_1 \wedge \cdots \wedge p_n$	$\forall x.([\bigwedge_{i=1}^{n} p_i \Uparrow \mathbf{t}] \vee [\bigvee_{i=1}^{n} p_i \Uparrow \mathbf{f}])$	$\forall x.(\neg[\bigwedge_{i=1}^{n} p_i \Uparrow \mathbf{t}] \vee \neg[\bigvee_{i=1}^{n} p_i \Uparrow \mathbf{f}])$

Figure 8. A policy in the first column is gap-free in an access-control state $\mathcal{M}$ iff $\mathcal{M}$ satisfies the constraint given in the second column, and similarly for conflict-freedom.

$$\frac{}{\mathsf{CF}(b \text{ if } ap_i)} \qquad \frac{\mathsf{CF}(p)}{\mathsf{CF}(\neg p)} \qquad \frac{\mathsf{CF}(p)\ \ \mathsf{CF}(q)}{\mathsf{CF}(p \wedge q)}$$

$$\frac{\mathsf{CF}(q)}{\mathsf{CF}(p \supset q)} \qquad \frac{\mathsf{CF}(q)}{\mathsf{CF}(p[\top \mapsto q])} \qquad \frac{\mathsf{CF}(p)\ \ \mathsf{CF}(q)}{\mathsf{CF}(p[\bot \mapsto q])}$$

Figure 9. Type system: judgment $\mathsf{CF}(p)$ asserts conflict-freedom of policy p.

$$\frac{}{\mathsf{GF}(b \text{ if true})} \qquad \frac{\mathsf{GF}(p)}{\mathsf{GF}(\neg p)} \qquad \frac{\mathsf{GF}(p)\ \ \mathsf{GF}(q)}{\mathsf{GF}(p \wedge q)}$$

$$\frac{\mathsf{GF}(q)}{\mathsf{GF}(p \supset q)} \qquad \frac{\mathsf{GF}(p)}{\mathsf{GF}(p \oplus q)} \qquad \frac{\mathsf{GF}(q)}{\mathsf{GF}(p \oplus q)}$$

$$\frac{\mathsf{GF}(p)\ \ \mathsf{GF}(q)}{\mathsf{GF}(p[\top \mapsto q])} \qquad \frac{\mathsf{GF}(q)}{\mathsf{GF}(p[\bot \mapsto q])}$$

Figure 10. Type system: judgment $\mathsf{GF}(p)$ asserts gap-freedom of policy p.

A firewall policy is interpreted on an access-control state

$$\mathcal{M} = (A_{\mathcal{M}}, \{ap_{i,j}^{\mathcal{M}} \mid 1 \le i \le n, 1 \le j \le 5\})$$

where the set $A_{\mathcal{M}}$ of accesses is all 5-tuples of the form explained above, and, for each i, the predicate $ap_{i,1}^{\mathcal{M}}$ returns true for those accesses whose host IP address is within a specified set of IP addresses $I_{i,1}$. For $j = 2, 3, 4$ the policies b_i if $ap_{i,j}$ are defined in the same manner through specified sets $I_{i,j}$ of IP addresses (for $j = 2$) and port numbers (for $j = 3, 4$). Access predicate $ap_{i,5}$ returns true for those accesses a whose fifth component service is in a specified set $Service_i$ of services. For simplicity we assume that applications cannot modify any predicate $ap_{i,j}$, and so executions of access requests will preserve access-control state $\mathcal{M}$. By Theorem 4, each rule r_i is conflict-free.

Note that in a "granting rule" r_i, in which $b_i = \mathbf{t}$, we have that $r_i(a) \ge_k \mathbf{f}$ is always false, and $r_i(a) \ge_k \mathbf{t}$ is true iff $ap_{i,j}^{\mathcal{M}}(a)$ holds for all $j = 1, \ldots, 5$. Similarly, in a "denying rule" r_i, in which $b_i = \mathbf{f}$, we have that $r_i(a) \ge_k \mathbf{t}$ is always false, and $r_i(a) \ge_k \mathbf{f}$ is true iff $ap_{i,j}^{\mathcal{M}}(a)$ holds for *some* $j = 1, \ldots, 5$.

We now ask how the rules $r_1, \ldots, r_n$ can be composed to form a firewall policy p_{fw}. In other words, what is the intended interpretation of the interaction between the rules?

Priorities. One interpretation is that "the first applicable rule wins", written as $p_{fw} = r_1 > \cdots > r_n$ in PBel. By Theorem 4, p_{fw} is conflict-free. Gap analysis is also important for firewall policies, as it can detect that important access requests are unspecified. Using the type system of Fig. 10 and Theorem 5 requires proof that r_n is gap-free, which is unlikely. But from Fig. 8 and Prop. 2 we learn that p_{fw} is gap-free in access-control state $\mathcal{M}$ just if $\mathcal{M}$ satisfies $\forall x. \bigvee_{i=1}^{n}(r_i \Uparrow \mathbf{t} \vee r_i \Uparrow \mathbf{f})$. This confirms our intuition that p_{fw} contains a gap for exactly those accesses that are gaps for all rules r_i.

Non-determinism. Another common interpretation of a collection of policy rules is that "any applicable rule may win". This amounts to a non-deterministic choice of applicable rules, which we may express as $p_{fw} = r_1 \oplus \ldots \oplus r_n$ in PBel. Ignoring policy conflicts for a moment, p_{fw} will deny (resp. grant) access a iff some r_i denies (resp. grants) access a. In particular, p_{fw} will in general not be conflict-free. For gap analysis, the type system for $\mathsf{GF}(p_{fw})$ can only help if some r_i is gap-free, which is unlikely. We therefore use our symbolic optimizations for gap and conflict analysis.

For conflict analysis, by Prop. 2 p_{fw} is conflict-free at access-control state $\mathcal{M}$ just if $\mathcal{M}$ satisfies constraint $\forall x.(\neg[\bigvee_{i=1}^{n} r_i \Uparrow \mathbf{t}] \vee \neg[\bigvee_{i=1}^{n} r_i \Uparrow \mathbf{f}])$. In other words, p_{fw} is *not* conflict-free in $\mathcal{M}$ iff $\mathcal{M} \models \exists x.([\bigvee_{i=1}^{n}(r_i \Uparrow \mathbf{f})] \wedge [\bigvee_{i=1}^{n}(r_i \Uparrow \mathbf{t})])$. Since each r_i is conflict-free, the latter holds iff there are distinct rules r_k, r_l and an access a in $\mathcal{M}$ with $[\![r_k]\!]_{\mathcal{M}}(a) \ge_k \mathbf{f}$ and $[\![r_l]\!]_{\mathcal{M}}(a) \ge_k \mathbf{t}$. From (2) we thus infer that conflicts occur at exactly those accesses for

which some rule grants all five tuples, and some other rule denies one of these five tuples.

For gap analysis, by Prop. 2 p_{fw} is gap-free at an access-control state $\mathcal{M}$ just if $\mathcal{M}$ satisfies $\forall x.\, \bigvee_{i=1}^{n}(r_i \Uparrow \mathbf{t} \vee r_i \Uparrow \mathbf{f})$. So p_{fw} has a gap at access a iff all rules r_i have a gap at a.

Logical Meet. Finally, a legitimate but impractical interpretation is that "all applicable rules win", which we may express as $p_{fw} = r_1 \wedge \cdots \wedge r_n$. As noted already, expression ($\mathbf{t}$ if ap_i) $\wedge$ ($\mathbf{f}$ if ap_j) is semantically equivalent to $\mathbf{f}$ if ap_j. So any firewall policy obtained by composing rules with $\wedge$ is equivalent to a policy whose results are contained in $\{\mathbf{t}, \bot\}$ or contained in $\{\mathbf{f}, \bot\}$. Such policies are clearly conflict-free. As this composition of p_{fw} is rather esoteric, we omit its gap analysis.

In [7], Capretta *et al.* present a conflict-detection method for Cisco IOS firewall policies. Using the Coq proof assistant, they define conflict in terms of the structure of access requests and policy rules in IOS firewall policies. They implement a conflict-detection algorithm in Coq's functional programming language and prove it correct, and then extract a correct OCaml program from the Coq program. The efficiency of the OCaml program is demonstrated by using it to detect conflicts in synthesized firewall policies having tens of thousands of rules.

The aims and results of our work are quite different. The work of Capretta *et al.* is targeted specifically to Cisco firewall policies: the rules of these policies are directly formalized, no explicit policy combinators are defined, and conflict is defined in terms of Cisco policy rules. Our work is not tied to Cisco's policies – we have a general-purpose policy language with explicit policy combinators, a general-purpose query language for expressing policy properties, and, through access predicates, a means to avoid tying the language and policy analysis to the particular structure of access requests. We have shown how Cisco firewall policy rules can be encoded in PBel, how the rules can be composed using PBel combinators, and that our standard notion of conflict captures the desired property. However, unlike Capretta *et al.*, we have not yet implemented our policy analyses, and so have not checked any firewall policies, real or synthesized.

A key difference between the work in [7] and our own is in the treatment of conflicts. By supporting the value $\top$ as a possible policy result, we can define conflict-freedom as a simple semantic property of policies. If $\top$ is not supported, then conflict freedom must be defined in terms of policy composition: a conflict exists if, on some access, one sub-policy produces result $\mathbf{t}$ while another produces result $\mathbf{f}$. However, this condition does not represent genuine conflict if the sub-policies are composed through an operator that resolves conflict. For example, in [7], conflict is defined as the existence of disagreeing firewall policy rules, but rule composition in firewall policies is a conflict-resolving priority composition operation. This means that the conflicts discovered in [7] are not genuine conflicts, although some of them may reflect unintended interactions between rules (this point is clearly discussed in that paper). We argue that such complications, which become more serious in a policy language with multiple composition operators, are avoided by using the Belnap bilattice as a policy result space.

7 Assume-guarantee reasoning

A query holds in a single access-control state $\mathcal{M}$ if $\mathcal{M}$ satisfies the constraint derived from the query, and a query holds of every access-control state if the constraint derived from the query is valid. We now present an assume-guarantee reasoning method that can be used to show that a query holds in a specified *class* of access-control states.

We take an assumption α to be a constraint that describes the class of access-control states we wish to consider, and that has x as its only (and free) variable. For example if α is $\mathtt{tt}$, then we want to reason about *all* possible access-control states. If α is $ap_1(x) \rightarrow ap_2(x)$, then we may only want to reason about access-control states $\mathcal{M}$ for which $ap_1^{\mathcal{M}} \subseteq ap_2^{\mathcal{M}}$. We take a guarantee to be a constraint c – in practice this will be the constraint derived from a query. For example, constraint c might be $\forall x.\, [ap_1(x) \vee ap_2(x)]$.

Definition 5 *An* assumption on x *is a quantifier-free constraint that either has no variable or has x as only variable.*

(Since an assumption α on x has at most free variable x, we can use the shorthand described earlier and write $\mathcal{M}, a \models \alpha$ instead of $\mathcal{M}, v[x := a] \models \alpha$.)

The idea of the proof method is that if $\mathcal{M}$ meets assumption α, and a prescribed condition links α and constraint c, then $\mathcal{M}$ will satisfy c. To make this precise we need to establish two things:

- How does access-control state $\mathcal{M}$ meet assumption α?
- What condition links assumption α and guarantee c?

An obvious starting point is to say that $\mathcal{M}$ meets assumption α if $\mathcal{M} \models \forall x.\alpha$, and that the required link between α and c is some kind of implication. Suppose we have a constraint of the form $\forall x.c'$. If $\mathcal{M} \models \forall x.\alpha$, and $\alpha \rightarrow c'$ is a tautology (understanding predicates $ap_i(x)$ in α and c' as atomic propositions), then one can derive $\mathcal{M} \models \forall x.c'$.

If the constraint of interest has form $\exists x.c'$, then $\mathcal{M} \models \exists x.c'$ can be derived from the fact that $\alpha \rightarrow c'$ is a tautology only from $\mathcal{M} \models \exists x.\, \alpha$ (and not from $\mathcal{M} \models \forall x.\, \alpha$ instead).

However, for this existential constraint $\exists x.c'$ we can get by with a much weaker set of conditions. Take the required condition linking α and the constraint to be that $\alpha \wedge c'$ is satisfiable. What it means for $\mathcal{M}$ to meet assumption α will

$$
\begin{array}{rcl}
(\!|\, \alpha \,\mathsf{impl}\, \mathsf{ff} \,|\!) & \textit{Iff} & \mathsf{false} \\
(\!|\, \alpha \,\mathsf{impl}\, \mathsf{tt} \,|\!) & \textit{Iff} & \mathsf{true} \\
(\!|\, \alpha \,\mathsf{impl}\, c_1 \wedge c_2 \,|\!) & \textit{Iff} & (\!|\, \alpha \,\mathsf{impl}\, c_1 \,|\!) \text{ and } (\!|\, \alpha \,\mathsf{impl}\, c_2 \,|\!) \\
(\!|\, \alpha \,\mathsf{impl}\, c_1 \vee c_2 \,|\!) & \textit{Iff} & (\!|\, \alpha \,\mathsf{impl}\, c_1 \,|\!) \text{ or } (\!|\, \alpha \,\mathsf{impl}\, c_2 \,|\!) \\
(\!|\, \alpha \,\mathsf{impl}\, \forall x.\, c' \,|\!) & \textit{Iff} & \alpha \rightarrow c' \text{ is a tautology} \\
(\!|\, \alpha \,\mathsf{impl}\, \exists x.\, c' \,|\!) & \textit{Iff} & \alpha \wedge c' \text{ is satisfiable}
\end{array}
$$

Figure 11. Definition of $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!)$ where α is an assumption on x and c a nesting-free constraint in negation normal form.

become more involved. We will say that $\mathcal{M}$ meets α if, whenever some $\mathcal{M}'$ satisfies α at access a', then there is an a in $A_{\mathcal{M}}$ such that the values of access predicates of $\mathcal{M}'$ at a' match the values of access predicates of $\mathcal{M}$ at a, for all access predicates appearing in α and c'.

Let us sketch how these conditions support assume-guarantee reasoning for constraint $\exists x.c'$. Assume $\alpha \wedge c'$ is satisfiable, which means there is some access-control state $\mathcal{M}'$ for which $\mathcal{M}' \models \exists x.(\alpha \wedge c')$, or equivalently there is some $\mathcal{M}'$ and access a' such that $\mathcal{M}', a' \models \alpha \wedge c'$. Because $\mathcal{M}'$ meets assumption α, we have (using the notion of "meets" just explained) that there is an access a such that $\mathcal{M}, a \models \alpha \wedge c'$, and therefore $\mathcal{M} \models \exists x.(\alpha \wedge c')$, which implies $\mathcal{M} \models \exists x.c'$ as desired.

We now work through the details, considering first the required condition linking assumption α and constraint c. An assumption on x can be interpreted as a propositional formula by treating each atom $ap_i(x)$ as an atomic proposition, and using a propositional valuation pv to interpret each $ap_i(x)$ as true or false. For each access-control state $\mathcal{M}$, and access a in $A_{\mathcal{M}}$, we define propositional valuation $pv(\mathcal{M}, a)$ by $pv(\mathcal{M}, a)(ap_i(x))$ is true iff $a \in ap_i^{\mathcal{M}}$. Then the *propositional interpretation* of an assumption α on x in access-control state $\mathcal{M}$ and valuation v is obtained by interpreting the Boolean connectives in the usual way, and interpreting atoms with the propositional valuation $pv(\mathcal{M}, v(x))$. A *tautology* is a valid propositional formula.

The condition linking assumption α and constraint c that we need for assume-guarantee reasoning is captured with notation $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!)$ (read as "α implies c"). This condition varies according to the structure of c (which we assume to be in negation normal form for sake of simplicity) and is defined in Fig. 11. That figure shows no cases for c of the form $ap_i(x)$ or $\neg c'$ since c and so its negation normal form are closed. We reduce computation of $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!)$ to SAT solving:

Proposition 3 *An assumption α on x interpreted propositionally is a tautology iff $\forall x.\, \alpha$ is valid.*

Thus, the value of $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!)$ is the result of combining the results of various calls to SAT solvers based on α and subformulas of c. This use of SAT solvers is possible since all expressions of the form $\alpha \rightarrow c'$ and $\alpha \wedge c'$ are by definition assumptions on x, and so Prop. 3 applies here. By putting an ordering on all $k \geq 0$ access predicate symbols occurring in $\alpha \wedge c$ we may alternatively compute $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!)$ through the synthesis of BDDs, whose size is at worst exponential in k [6] and so not directly dependent on the size of $\alpha \wedge c$.

Example 7 *Let assumption α be $\neg ap_1(x)$, as computed in Example 2, and let constraint c be $[\![\, \phi \,]\!]$ for ϕ being $(\mathtt{t} \text{ if } ap_1) \wedge (\mathtt{f} \text{ if } ap_2) \leq_t (\mathtt{f} \text{ if } ap_2) \wedge \neg(\mathtt{t} \text{ if } ap_1)$. Then $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!)$ iff $\neg ap_1(x) \rightarrow (ap_1(x) \rightarrow ap_2(x))$ is a tautology, which is the case. Thus $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!) = true$. This also illustrates that the constraints generated from a query may serve as assumptions for the evaluation of other queries.*

As explained earlier, we use two different notions of an access-control state $\mathcal{M}$ "meeting" an assumption α.

Definition 6 *Let $\mathcal{M}$ be an access-control state, α an assumption on x, and c a nesting-free constraint. Then*

- *α is* $\mathcal{M}$-valid *if $\mathcal{M} \models \forall x.\, \alpha$*
- *α is* $\mathcal{M}$-complete for *c if, for all access-control states $\mathcal{M}'$, assumptions β on x whose access predicate symbols all occur in c, and accesses a' in $\mathcal{M}'$ with $\mathcal{M}', a' \models \alpha \wedge \beta$, there is an access a in $\mathcal{M}$ such that $\mathcal{M}, a \models \alpha \wedge \beta$.*

Theorem 6 *Let $\mathcal{M}$ be an access-control state, α be an $\mathcal{M}$-valid assumption, and c be a nesting-free constraint whose negation normal form contains no existential quantification. Then $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!)$ implies $\mathcal{M} \models c$.*

Showing $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!)$ is not a complete method for showing that constraint c holds. For access-control state $\mathcal{M}$ with set of accesses $A_{\mathcal{M}} = \{0, 1\}$, $ap_1^{\mathcal{M}} = \{0\}$, $ap_2^{\mathcal{M}} = \{0, 1\}$, $ap_3^{\mathcal{M}} = \{\}$, and $c = \forall x.\, [ap_1(x) \rightarrow ap_2(x)]$ we have $\mathcal{M} \models c$ but $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!)$ is false for $\alpha = \neg ap_3(x)$.

Example 8 *We revisit Example 7. The query ϕ has form $p \leq_t q$ and so is negation-free. Thus we may apply Theorem 6 for various choices of assumption α.*

- *For $\alpha = \neg ap_1(x)$, which says that access predicate ap_1 is empty, we saw in Example 7 that $(\!|\, \alpha \,\mathsf{impl}\, [\![\, \phi \,]\!] \,|\!)$ is true, and so $\mathcal{M} \models \phi$ holds for all $\mathcal{M}$ with $\mathcal{M} \models \forall x.\, \neg ap_1(x)$ by Theorems 6 and 3.*
- *For $\alpha = ap_2(x) \rightarrow \neg ap_1(x)$, we have $(\!|\, \alpha \,\mathsf{impl}\, [\![\, \phi \,]\!] \,|\!)$ is false as $\forall x.\, [ap_2(x) \rightarrow \neg ap_1(x)] \rightarrow [ap_1(x) \rightarrow ap_2(x)]$ isn't valid. So we don't know if $\mathcal{M} \models \phi$.*

Theorem 7 *Let $\mathcal{M}$ be an access-control state, α be an assumption that is $\mathcal{M}$-complete for c, and c be a nesting-free constraint whose negation normal form contains no universal quantification. Then $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!)$ implies $\mathcal{M} \models c$.*

Combining these assume-guarantee results, we obtain their straightforward generalization for nesting-free constraints c with both universal and existential quantification.

Theorem 8 *Let $\mathcal{M}$ be an access-control state, α be an $\mathcal{M}$-valid and $\mathcal{M}$-complete assumption on x, and c be a nesting-free constraint. Then $(\!|\, \alpha \,\mathsf{impl}\, c \,|\!)$ implies $\mathcal{M} \models c$.*

For any query ϕ, constraint $[\![\ \phi\ ]\!]$ is nesting-free. Therefore we can combine Theorem 8 with Theorem 3 to support assume-guarantee reasoning directly on queries.

Corollary 1 *Let $\mathcal{M}$ be an access-control state, ϕ be a query, and α be an $\mathcal{M}$-valid and $\mathcal{M}$-complete assumption. Then $(\!|\, \alpha \,\mathsf{impl}\, [\![\ \phi\ ]\!] \,|\!)$ implies $\mathcal{M} \models \phi$.*

Theorem 7 suggests that assumptions on x need to encode a sufficient amount of information about $\mathcal{M}$ so that they are $\mathcal{M}$-complete for c, since our constraint-based analysis (the satisfiability check of $\alpha \wedge c'$) then never produces "spurious" witnesses to the satisfiability of $\alpha \wedge c'$ that are not such witnesses in $\mathcal{M}$. So even if c contains no existential quantifications in negation normal form, knowing that α is $\mathcal{M}$-complete for c means that all satisfiability witnesses of $\alpha \wedge c'$ are genuine counter-examples for $\mathcal{M} \models \forall x.\, c'$.

Before giving an example, we first show that one can always synthesize assumptions on x that are $\mathcal{M}$-valid and $\mathcal{M}$-complete for c over a finite set of accesses.

Theorem 9 *Let $\mathcal{M} = (A_{\mathcal{M}}, \{ap_i^{\mathcal{M}} \mid i \geq 1\})$ be an access-control state with finite set $A_{\mathcal{M}}$. For any nesting-free constraint c with set of atoms Γ there is an assumption $\alpha = \bigvee_{a \in A_{\mathcal{M}}} m_a$ on x with*

$$m_a = \bigwedge_{ap_i \in \Gamma,\, a \in ap_i^{\mathcal{M}}} ap_i(x) \wedge \bigwedge_{ap_i \in \Gamma,\, a \notin ap_i^{\mathcal{M}}} \neg ap_i(x)$$

that is $\mathcal{M}$-valid and $\mathcal{M}$-complete for c.

Example 9 *From an access-control state $\mathcal{M}$ (and a query ϕ) we derive an $\mathcal{M}$-complete assumption α using Theorem 9. Let query ϕ be $\neg(p \leq_t q)$, where $p = (\mathbf{f}\ \mathsf{if}\ ap_1) \wedge (\mathbf{t}\ \mathsf{if}\ ap_2)$ and $q = (\mathbf{t}\ \mathsf{if}\ ap_2) \oplus (\mathbf{t}\ \mathsf{if}\ ap_1)$. Then constraint c is $[\![\ \phi\ ]\!] = \exists x. \neg[(ap_1(x) \to \mathsf{ff}) \wedge (\mathsf{ff} \to ap_2(x) \wedge ap_1(x))]$, which simplifies to $\exists x.\, ap_1(x)$.*

Let $\mathcal{M}$ be an access-control state with $\{\} \neq ap_1^{\mathcal{M}} \neq A_{\mathcal{M}}$. For each $a \in A_{\mathcal{M}}$, monomial m_a from Theorem 9 is either $ap_1(x)$ or $\neg ap_1(x)$, and both monomials occur in $\alpha = \bigvee_{a \in A_{\mathcal{M}}} m_a$ as $\emptyset \neq ap_1^{\mathcal{M}} \neq A_{\mathcal{M}}$. So α is equivalent to tt. *Then $(\!|\, \mathsf{tt} \,\mathsf{impl}\, \exists x.\, ap_1(x) \,|\!)$ iff $\mathsf{tt} \wedge ap_1(x)$ is satisfiable iff $\neg ap_1(x)$ is not a tautology (which is the case). Any witness for the falsity of $\neg ap_1(x)$ has to make $ap_1(x)$ true. But that propositional valuation has form $pv(\mathcal{M}, a)$ as $ap_i^{\mathcal{M}} \neq \emptyset$.*

Therefore α, $\mathcal{M}$, and c satisfy the assumptions of Theorem 7, so we infer from Theorems 7 and 3 that $\mathcal{M} \models \phi$.

8 Expressiveness

An access-control policy language will ideally be able to capture every policy as a language expression. That is to say, for any access-control state $\mathcal{M}$ and any function $f\colon A_{\mathcal{M}} \to \mathbf{Four}$ there should be some policy expression p with $[\![p]\!]_{\mathcal{M}} = f$. A policy language is *functionally complete* if this holds.

PBel is not functionally complete. Let $A_{\mathcal{M}} = \mathbf{Four}$, $f(x) = x$ for all $x \in \mathbf{Four}$ and $ap_i^{\mathcal{M}} = \{\mathbf{t}, \top\}$ for all $i \geq 1$. There cannot be a PBel expression p with $[\![p]\!]_{\mathcal{M}} = f$. This is shown by structural induction on p noting that f behaves differently for $\mathbf{t}$ and $\top$ but both elements satisfy the same set of access predicates in $\mathcal{M}$.

This raises the question of whether functions f that identify accesses that are not distinguishable by access predicates are those functions expressible as PBel expressions.

Definition 7 *For access-control state $\mathcal{M}$ we define an equivalence relation $\equiv_{\mathcal{M}}$ on $A_{\mathcal{M}}$ by*

$$a \equiv_{\mathcal{M}} a' \ \textit{Iff}\ (\forall i \geq 1\colon a \in ap_i^{\mathcal{M}}\ \textit{Iff}\ a' \in ap_i^{\mathcal{M}})$$

We call a function $f\colon A_{\mathcal{M}} \to \mathbf{Four}$ data-independent in $\mathcal{M}$ *iff $f(a) = f(a')$ whenever $a \equiv_{\mathcal{M}} a'$.*

Data-independence [17] exploits that many systems, e.g. security protocols [23], treat data of the same type in the same manner. In our context it means a function f is independent of the actual data (here accesses), but only dependent on their types (here access predicates with "negation" and "intersection" types). The semantics $[\![p]\!]_{\mathcal{M}}$ of all PBel expressions is data-independent in $\mathcal{M}$.

Proposition 4 *Let $\mathcal{M}$ be an access-control state and p be a PBel expression. Then $[\![p]\!]_{\mathcal{M}}\colon A_{\mathcal{M}} \to \mathbf{Four}$ is data-independent in $\mathcal{M}$.*

We show that PBel can express all data-independent functions in $\mathcal{M}$. Let $(\mathbf{t}\ \mathsf{if}\ \neg ap_i)$ be a shorthand for the PBel expression

$$\neg[(\mathbf{t}\ \mathsf{if}\ ap_i)[\bot \mapsto \mathbf{f}] \wedge (\mathbf{t}\ \mathsf{if}\ \mathsf{false})]$$

Proposition 5 *Let $\mathcal{M}$ be an access-control state. Then*

$$[\![\mathbf{t}\ \mathsf{if}\ \neg ap_i]\!]_{\mathcal{M}}(a) \quad = \quad \begin{cases} \mathbf{t} & \textit{if } a \notin ap_i^{\mathcal{M}} \\ \bot & \textit{otherwise} \end{cases}$$

Below we write $\sum$ for the $n$ary versions of $\oplus$, and $\bigwedge$ for $n$ary versions of $\wedge$.

Theorem 10 *Let $\mathcal{M}$ be an access-control state with finite set $A_{\mathcal{M}}$ of accesses, and let $f\colon A_{\mathcal{M}} \to \mathbf{Four}$ be a function that is data-independent in $\mathcal{M}$. Then f equals $[\![p]\!]_{\mathcal{M}}$ for the PBel expression $p = p_{\mathbf{t}} \oplus p_{\mathbf{f}}$ where*

$$\begin{aligned} p_{\mathbf{t}} &= \sum_{a \in A_{\mathcal{M}} \mid f(a) \geq_k \mathbf{t}} p_a \\ p_{\mathbf{f}} &= \sum_{a \in A_{\mathcal{M}} \mid f(a) \geq_k \mathbf{f}} \neg p_a \\ p_a &= \Big(\bigwedge_{i \mid a \in ap_i^{\mathcal{M}}} \mathbf{t} \text{ if } ap_i \Big) \wedge \Big(\bigwedge_{i \mid a \notin ap_i^{\mathcal{M}}} \mathbf{t} \text{ if } !ap_i \Big) \end{aligned}$$

Thus PBel can express exactly those functions $f\colon A_{\mathcal{M}} \to \mathbf{Four}$ that, for some access-control state $\mathcal{M}$, are data-independent in $\mathcal{M}$. Also, the clauses for $p \supset q$ and for $p[\top \mapsto q]$ in PBel can be expressed by a composition of other clauses in that language as these two clauses were not used in the proofs of Prop. 5 and Theorem 10. Moreover, a replacement of all $\oplus$ and their $n$ary versions in Theorem 10 with priority composition $>$ expresses conflict-freedom:

Theorem 11 *PBel, as shown in Fig. 2, but without clause $p \oplus q$, expresses exactly the functions $f\colon A_{\mathcal{M}} \to \mathbf{Four}$ that are data-independent in $\mathcal{M}$ and conflict-free.*

In summary, we have functional completeness for the level of abstraction, the access predicates ap_i, that we consider. In future work we mean to investigate functional completeness for the richer types

$$(A_{\mathcal{M}} \to \mathbf{Four})^k \to (A_{\mathcal{M}} \to \mathbf{Four})$$

to understand better the expressiveness of PBel for higher-order composition operators. Although operator $\supset$ of PBel has not been used in our examples, we suspect it is needed for securing this functional completeness result, which is "polymorphic" in access-control states $\mathcal{M}$.

9 Related Work

The idea that policies can produce results other than "grant" and "deny" is familiar, but surprisingly little has been done in the area of multi-valued policy algebras. Policy formalisms that capture "grant", "deny", and "undefined" result values include the default logic-based formalism of Woo and Lam [25], and SPL [22]. A formalism that captures "grant", "deny", and "conflict" can be found in [19], where the outcomes depend on the provability of formulas in defeasible logic. XACML [9] uses values that correspond to "grant", "deny", and "undefined", plus a value "indeterminate" that reflects a processing error.

SPL is one of the few policy languages that uses a multi-valued algebra for policy composition. SPL policies can be composed using a three-valued logic, with operators AND, OR, NOT, plus indexed forms of AND and OR. This logic is unlike Kleene's strong three-valued logic [16] and every other standard three-valued logic we know. In PBel, the SPL policy composition p AND q can be written $(p \oplus q)[\top \mapsto \mathbf{f}]$ and composition p OR q can be written $(p \oplus q)[\top \mapsto \mathbf{t}]$. We believe these encodings demonstrate that policy composition is simpler when composition operators are not required to resolve conflicts. SPL's basic rule domain :: decide can also be encoded in PBel. For that we use the fact, not shown in this extended abstract, that expressions of the form $(b \text{ if } \tau)$ can be translated into PBel whenever τ is a Boolean combination of access predicates. The encoding of SPL's rule above is then $(\mathbf{t} \text{ if } \mathit{domain} \wedge \mathit{decide}) \oplus (\mathbf{f} \text{ if } \mathit{domain} \wedge \neg \mathit{decide})$.

XACML also has multi-valued operations for composition, but separate ones for rule composition and policy composition. We consider here XACML's four main policy composition operations (called "policy-combining algorithms"). Although defined on non-empty policy sets, they can be expressed as commutative and associative binary policy operations. If XACML's "indeterminate" is understood as $\top$, then XACML's "permit-overrides" algorithm on policies p and q can be written in PBel as $(p+q)[\top \mapsto \mathbf{f}]$, its "first-applicable" algorithm can be written $p > q$, and its "only-one-applicable" algorithm can be written $(p \oplus q) \oplus ((p \oplus \neg p) \otimes (q \oplus \neg q))$. However, it seems more likely that XACML's "indeterminate" should sometimes be treated as $\bot$ and sometimes as $\top$, depending on circumstance.

We briefly mention one other policy algebra, which is not however multi-valued. In [3], Bonatti *et al.* interpret a policy expression as a set of accesses, and then define composition operators on these access sets. For example, expression $p\&q$ denotes the intersection of the access sets denoted by p and q, and $p + q$ their union. This approach is simple, but does not allow for the distinction between an access that should be denied and an access that is outside the scope of a policy. Negative permissions are achieved through a set difference operator $p - q$. This operation gives denials modelled by q higher priority than permissions modelled by p, so conflicts cannot arise. Furthermore, one cannot symmetrically state that grants modelled by one policy should take priority over the denials modelled by another.

Composition in some policy languages is a syntactic operation. For example, a form of policy inheritance is supported in the Cisco Management Center for Firewalls [8]. A hierarchy of pairs of access lists is created; a firewall policy is assembled by forming a single access list from the pairs along a path in the hierarchy from a leaf to the root. Another example is in the policy language of Lee *et al.* [18].

Here a policy is a pair of theories of defeasible logic, each theory consisting of three kinds of rules, plus a rule ordering. These rules, as in default logic [21], allow tentative or definite conclusions to be inferred. Composition takes an ordered set of policies and produces a single policy by unioning the rules of defeasible theories and updating the ordering among the rules of each theory.

The material in Sections 2 and 3 of this paper summarize the main content of [5]. The version of PBel presented here has two operators not found in the language of [5]: Nondeterministic Choice, and Refinement. This is merely for convenience; the additional operators do not add expressive power. Only a single formal result on expressiveness is found in [5], stating that the policy language is powerful enough to express every mapping from accesses to **Four** if strong-enough access predicates are available. In Section 8 of this paper we have shown the more general result that PBel expresses exactly the data-independent mappings from accesses to **Four**, and further that PBel without $\oplus$ expresses exactly the data-independent and conflict-free mappings from accesses to **Four**. No material on policy analysis (sections 4 – 7 of this paper) is contained in [5].

10 Future work

In [25], Woo and Lam list as a requirement that a policy language should support "closure". Informally, closure allows the result of a policy on an access to depend on the result of a sub-policy on *another* access. For example, a policy that regulates read and write permissions to documents may have to be extended so that a document is readable by a user just if it is writable.

In logic-based policy languages, the ability to compose policies in this way is supported by inference rules such as

$$grant(\mathsf{user}, \mathsf{file}, \mathsf{read}) \text{ :- } grant(\mathsf{user}, \mathsf{file}, \mathsf{write})$$

This rule has two salient features: the grant predicate appears as both a premise and a conclusion, and the access mentioned in the premise differs from the access in the conclusion. The difficulty in expressing this idea in PBel has to do with this latter feature. Our definition of $[\![p]\!]_{\mathcal{M}}(a)$ means that a policy p on an access a must be a function of the results of its sub-policies *on the same access* a, not on some other access (ignoring the effect of Prop. 4).

To accommodate this feature, we add a clause $p\langle f\rangle$ to PBel, where unary function symbol f denotes an "access mapping". Its semantics in access-control state $\mathcal{M}$ is a total function $f^{\mathcal{M}}\colon A_{\mathcal{M}} \to A_{\mathcal{M}}$ and so we may extend our denotational semantics with a clause

$$[\![p\langle f\rangle]\!]_{\mathcal{M}}(a) = [\![p]\!]_{\mathcal{M}}(f^{\mathcal{M}}(a))$$

If accesses have the form $\langle \mathsf{user}, \mathsf{action}, \mathsf{document}\rangle$ and if mapping $ReadToWrite$ is interpreted as a function that maps every access $\langle \mathsf{u}, \mathsf{read}, \mathsf{d}\rangle$ to $\langle \mathsf{u}, \mathsf{write}, \mathsf{d}\rangle$ and leaves all other accesses fixed, we can write $p\langle ReadToWrite\rangle > p$ for a policy that grants a read access whenever the corresponding write access was granted in p – regardless of p's result on the access. In this way PBel can be extended to relate permissions of one policy to permissions on other accesses in another policy. The constraints that are generated then also contain terms built from variable x and function symbols but can be embedded into a decidable fragment of first-order logic [4], e.g. into the fragment $[\exists^*\forall\exists^*]$ (all, all). We mean to identify optimal decision procedures for the constraints that we generate by the addition of a $\langle f\rangle$ operator to PBel.

Because PBel does not support policy variables, one cannot write recursive policy rules. We therefore avoid the cost and difficulty that can arise with such rules. Support for bounded recursion would be of interest, e.g., in role-based access control [10, 11], where permissions and denials may be inherited across sub-roles. We mean to study to what extent PBel can already support role-based access control states (e.g., through assume-guarantee reasoning) or whether changes to PBel (e.g., by making $\wedge$ and $\oplus$ dependent on a relation symbol) would accommodate this important model (and perhaps even "closure") better.

11 Conclusions

We have presented various methods for the analysis of PBel policies. First we have shown how to translate policy queries to standard, two-valued logical formulas over access predicates, allowing the satisfaction of queries by policies to be checked using existing reasoning machinery such as BDDs, SAT solvers, and logical simplifiers. We have given static analyses for the common properties of conflict and gap-freedom. To support proving policy properties that are only expected to hold for certain classes of access requests, we have also devised a method for assume-guarantee reasoning about policies. Again, such reasoning can be achieved by means of SAT solvers or BDDs.

We have also formalized Cisco IOS policy analysis in PBel, and shown how the analysis of conflict and gap-freedom can be accomplished under various choices of rule composition.

We would like to reiterate two key elements of our work. By basing PBel on Belnap's four-valued logic, the properties of conflict-freedom and gap-freedom can be expressed as simple, purely semantic properties of policies. In contrast, two-valued policies cannot exhibit conflicts or gaps: basic policies cannot exhibit them, and conflicts will necessarily be resolved through policy composition. Therefore, conflict-freedom in two-valued policies must be expressed as the absence of disagreement between sub-policies. This is unsatisfactory, however, because disagreements between

sub-policies may or may not reflect real problems. It may be that these disagreements are anticipated and are resolved appropriately through composition. With PBel, sub-policies can either be composed in a way that resolves conflicts (when they are expected), or in a way that allows conflicts to propagate (when they are unexpected) so that problematic conflicts then can be detected and eliminated.

The second key element is the use of access predicates in PBel, which abstract away from the specifics of access requests and environmental data in an application. Access predicates serve as Boolean observables, and thus provide for generality of the language, for flexibility in the degree of granularity of access requests, and facilitate efficient analysis through the use of off-the-shelf SAT solvers or BDDs. Since BDDs, unlike SAT solvers, provide an implementation of Boolean functions – and so of PBel expressions – BDDs appear to be the method of choice for implementing both PBel and its analyses.

Acknowledgments

Glenn Bruns was supported in part by US National Science Foundation grant 0244901. This work has, in part, be performed in collaboration with the project "Aspects of Security for Citizens", funded by the Danish Strategic Research Council. Nir Piterman gave useful comments on drafts of this paper.

References

[1] O. Arieli and A. Avron. The value of the four values. *Artificial Intelligence*, 102(1):97–141, 1998.

[2] N. D. Belnap. A useful four-valued logic. In J. M. Dunn and G. Epstein, editors, *Modern Uses of Multiple-Valued Logic*, pages 8–37. D. Reidel, Dordrecht, 1977.

[3] 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.

[4] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer Verlag Berlin, 1997.

[5] G. Bruns, D. Dantas, and M. Huth. A simple and expressive semantic framework for policy composition in access control. In *Proc. of the 2007 ACM workshop on Formal Methods in Security Engineering (FMSE '07)*, 2007.

[6] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[7] V. Capretta, B. Stepien, A. Felty, and S. Matwin. Formal correctness of conflict detection for firewalls. In *Proc. of the 2007 ACM workshop on Formal Methods in Security Engineering (FMSE '07)*, pages 22–30, New York, NY, USA, 2007. ACM Press.

[8] CiscoWorks. *Using Management Center for Firewalls 1.3.2*. Cisco Systems, Inc., 2004.

[9] T. M. (editor). eXtensible Access Control Markup Language (XACML) Version 2.0. Committee specification, OASIS, February 2005.

[10] D. Ferraiolo and D. R. Kuhn. Role-Based Access Control. In *Proc. of the NIST-NSA National (USA) Computer Security Conference*, pages 554–563, 1992.

[11] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control (Second Edition)*. Artech House, Inc., Norwood, MA, USA, 2003.

[12] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *27th International Conference on Software Engineering (ICSE 2005)*, pages 196–205, 2005.

[13] M. Fitting. Bilattices are nice things. In T. Bolander, V. Hendricks, and S. A. Pedersen, editors, *Self-Reference*, pages 53–77. Center for the Study of Language and Information, 2006.

[14] M. Ginsberg. Multivalued logics: a uniform approach to reasoning in AI. *Computational Intelligence*, 4:256–316, 1988.

[15] J. Halpern and V. Weissman. Using first-order logic to reason about policies. In *Proceedings of the Computer Security Foundations Workshop (CSFW'03)*, 2003.

[16] S. C. Kleene. *Introduction to Metamathematics*. D. Van Nostrand, 1952.

[17] R. S. Lazic. *A Semantic Study of Data Independence with Applications to Model Checking*. PhD thesis, Computing Laboratory, Oxford University, April 1999.

[18] A. J. Lee, J. P. Boyer, L. E. Olson, and C. A. Gunter. Defeasible security policy composition for web services. In *FMSE '06: Proceedings of the fourth ACM workshop on Formal methods in security*, pages 45–54, New York, NY, USA, 2006. ACM Press.

[19] M. McDougall, R. Alur, and C. A. Gunter. A model-based approach to integrating security policies for embedded devices. In *Proceedings of Fourth ACM International Conference On Embedded Software (EMSOFT 2004)*, pages 211–219, 2004.

[20] D. of Defense of the United States of America. Trusted computer system evaluation criteria. DoD Standard 5200.28-STD.

[21] R. Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

[22] C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. SPL: An access control language for security policies and complex constraints. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2001)*, 2001.

[23] A. W. Roscoe. Proving security protocols with model checkers by data independence techniques. In *Proc. of the 11th IEEE Computer Security Foundations Workshop*, pages 84–95, June 1998.

[24] J. Sedayao. *Cisco IOS Access Lists*. O'Reilly, 2001.

[25] T. Y. C. Woo and S. S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2-3):107–136, 1993.