An Algebra for Enterprise Privacy Policies Closed Under Composition and Conjunction

Dominik Raub[1] and Rainer Steinwandt[2]

[1] Department of Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland
[2] Department of Mathematical Sciences, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431, USA

Abstract. A prerequisite for processing privacy-sensitive data with automatic tools is a fine-grained formalization of privacy policies along with appropriate operators to manipulate such policies. The most promising results for the formalization of privacy policies so far have been achieved with the language EPAL resp. its academic counterpart E-P3P.
As shown at ESORICS 2004, in the existing form E-P3P has fundamental limitations in the expressability of composed policies as desired in projects involving multiple departments or enterprises. We describe a Novel Algebraic Privacy Specification (NAPS) which addresses these problems by offering conjunction, composition and scoping operators, which are defined analogously to those known from E-P3P, but exhibit desirable algebraic properties. Most notably NAPS is, in contrast to E-P3P, closed under all of these operators. Also, we show how existing E-P3P policies fit into the NAPS framework.

1 Introduction

The processing of privacy-sensitive data is accompanied by increasingly complex regulations that have to be taken into account. Hence research on formal models for privacy policies and (semi-)automatic tools for processing these policies is gaining attention in academic and industrial research. Compared to access control, the available tools for managing privacy policies are far from being satisfactory, and even the formalization of privacy policies is not solved in a satisfactory manner yet. The association of purposes and obligations with data access, as needed for privacy policies, complicates the application of access control tools significantly and motivates the development of specific tools for expressing and processing privacy policies. Here it is useful to differentiate between i) the "simple" requirements needed to deal with privacy issues relating enterprise and private users and ii) the fine-grained tools needed for handling privacy-sensitive data in enterprise-to-enterprise relations. For handling privacy policies in the latter context, the currently most promising approach is the Enterprise Privacy Authorization Language (EPAL) resp. its academic abstraction E-P3P (see [1, 17, 5, 4]).

Having in mind large projects, possibly involving several enterprises, the question for a modular construction of privacy policies naturally arises. Based on EPAL resp. E-P3P an interesting first step in this direction has been presented by Backes et al. at ESORICS 2003 [5]. Albeit being of great value, the operator for ordered composition discussed therein does not offer the desired flexibility for composing privacy policies yet. Thus, motivated by well-known algebraic tools from access control [9, 24, 10, 25], at ESORICS 2004 Backes et al. put forward an *algebra for composing enterprise privacy policies* [4]. Next to the ordered composition of privacy policies formulated in E-P3P, here also conjunction and disjunction operators are introduced, thereby allowing for more flexibility in deriving new privacy policies from existing ones in a modular way. Unfortunately, the algebra in [4] suffers from fundamental limitations in the expressiveness of EPAL resp. E-P3P, which prohibit an intuitive definition of policy conjunction and disjunction as would be desirable. To cope with this problem, [4] introduces a class of *well-founded* privacy policies, for which a comparatively convenient algebraic treatment is possible. However, this policy class is not closed under ordered composition, which makes the combination of different policy operators quite inconvenient. Also, with regard to the operator semantics some aspects are not fully satisfactory at the moment. E. g., i) incorporation of the default ruling into the policy, as proposed before the definition of ordered composition in [4], makes ordered composition trivial for any but a “don’t care” default ruling. The default ruling of the first—higher priority—policy, now incorporated in the ruleset, will treat all queries, and leave none to be treated by the second—lower priority—policy; ii) the fact that in E-P3P obligations on, say, a department are always at least as strict as obligations imposed on members of the department makes the definition of minimum requirements on a department somewhat cumbersome; iii) restricting rulings to “allow”, “deny” and “don’t care” (plus obligations) as in E-P3P is not really suited to differentiate between “access can be allowed” and “access must not be denied”. When dealing with privacy-sensitive data, a convenient way to express these different types of “allow” is desirable: While accessing a data item for marketing purposes may be acceptable, legal regulations may impose that a client may never be denied access to her personal data.

The Novel Algebraic Pricavy Specification (NAPS) framework described below addresses these issues, thereby overcoming some limitations of E-P3P. The semantics of NAPS is more or less straightforward, defined by simply evaluating rules by descending priority. Definition of minimal requirements, say in the privacy policy of a company, is well supported by NAPS and the refinement of these to department or workgroup policies can be achieved through ordered composition. NAPS allows the separate specification of obligations both for the case when access to a data item is denied or granted. Furthermore NAPS is closed under its operators, including conjunction, ordered composition, and scoping. Also the handling of partial or missing information about privacy relevant data (such as age of a customer) is taken into account.

Despite these attractive features it would certainly not be justified to claim NAPS to be a totally superior substitute for E-P3P. E. g., so far no automatic tools for processing NAPS policies exist, while for E-P3P resp. EPAL progress in the development of such tools has been made already (cf. [3, 2]). Thus, much work remains to be done to explore the practical value of NAPS, but we think the results achieved so far certainly justify further research in this direction.

Further related work The starting point to develop NAPS was the E-P3P based algebra to compose enterprise privacy policies presented in [4] and building on [5]. In particular, NAPS keeps the established approaches to formulate purpose-specific requests and obligations associated with data accesses. Individual privacy-specific aspects of data processing have been discussed already in [15, 6, 8, 21], for instance. E-P3P resp. EPAL offers one of the most elaborated frameworks in this context and getting rid of limitations in policy composition of these frameworks without giving up E-P3P's resp. EPAL's merits is indeed desirable.

Policy composition has been explored in various contexts already (cf., e. g., [19, 23, 14, 12]), particularly in access control [11, 16, 9, 27, 10], and—as already mentioned in [4]—existing algebraic tools for access control [9, 10, 24, 25] are a key motivation to establish algebraic tools for the treatment of privacy policies.

2 Basic Definitions

Statements to be captured by privacy policies typically have a form like "*John Doe* from *sales* may *read customer data* for *marketing* purposes, if the *customer has consented*, with the *obligation to notify the customer*." To formalize such statements we follow the established approach of using hierarchies to represent users, data, actions, and purposes (cf. [5, 4]). Also for expressing obligations, we mainly adopt the modelling from [4]. For expressing conditions and rulings of policies, our approach deviates from E-P3P and we refer to Section 2.4 for a discussion of how to embed E-P3P policies into the NAPS framework.

2.1 Hierarchies, Obligations, and Conditions

As in E-P3P we use hierarchies to model users, data, purposes and actions. This in particular enables the specification of policies applying to entire subhierarchies, e. g. all users within ("$\leq$") the sales department. Unlike [5, 4] we do not require hierarchies to have unique predecessors:

Definition 1 (Hierarchy). *A* hierarchy $(H, \leq_H)$ *is an ordered*[3]*, finite set. A hierarchy* $(H, \leq_H)$ *is a* subhierarchy *of a hierarchy* $(G, \leq_G)$*, written* $(H, \leq_H) \subseteq (G, \leq_G)$*, iff* $(H \subseteq G)$ *and* $(\leq_H \subseteq \leq_G)$*. We set* $(H, \leq_H) \cup (G, \leq_G) := (H \cup G, (\leq_H \cup \leq_G)^*)$ *with* * *denoting reflexive, transitive closure. Note that* $(H, \leq_H) \cup (G, \leq_G)$ *is only a hierachy if* $(\leq_H \cup \leq_G)^*$ *is an order, in which case we call* H *and* G compatible*.*

A privacy policy not only regulates access to data, but can impose *obligations* like "delete this data set within two weeks" or "notify the customer". Analogously as in [4], it is convenient to impose some algebraic structure on the set of obligations:

Definition 2 (Obligation Model). *An* obligation model $(O, \leq, \wedge, \top, \bot)$ *is a meet-semilattice (a commutative, idempotent monoid) [7, 26] with maximal element* $\top$*, the empty obligation or no obligation, and minimal element* $\bot$*, the unfulfillable obligation. In keeping with the definition of a semilattice we have* $\forall o, p \in O : o \leq p :\iff o \wedge p = o$*.*

We assume that all occurring obligation models $(O, \leq)$ *are subsets of a fixed (super) obligation model* $(\mathcal{O}, \leq_{\mathcal{O}})$ *such that* $\leq$ *is the restriction of* $\leq_{\mathcal{O}}$ *to* $O \times O$*.*

[3] We use order and partial order synonymously. Total orders are explicitely called so.

Imposing the obligation $\bot$ indicates, that an action must not be performed. Imposing $\top$ signifies, that an action may be performed without restriction.

Remark 1 (Standard Obligation Model). For most purposes it will suffice to imagine a powerset lattice $(\mathfrak{P}(\tilde{O}), \supseteq)$ over a set of elementary obligations $\tilde{O}$ (like the ones stated above) with the set union $\cup$ as conjunction $\wedge$, $\top := \emptyset$, $\bot := \tilde{O}$ and for $A, B \in \mathfrak{P}(\tilde{O})$: $A \leq B \iff A \supseteq B$.

We want to enable policy specifications, making no final decision upon granting access or not, but still providing an obligation for either case. This allows to defer the final decision of granting or denying access to another policy or to an access control system (in the simplest case just granting (or denying) access by default). Therefore, we design NAPS rules to yield a pair of obligations $r = (o^+, o^-)$ as ruling, where o^+ is the obligation to be imposed if access to the data is later granted whereas o^- is the obligation to be imposed if access to the data is later denied. A rule may impose the obligation $o^+ = \bot$, meaning access must not be granted, or it may impose $o^- = \bot$, meaning access must not be denied. The ruling $o^+ = \bot$, $o^- = \bot$ is contradictory and used as an error state.

Definition 3 (Ruling). *Let O be an obligation model. A* ruling *r is a pair of obligations $r = (o^+, o^-) \in O \times O$ where o^+ is the obligation imposed on granting access to a data element and o^- is the obligation imposed on denying access to a data element. Defining the operation $\wedge$ elementwise on the pairs (o^+, o^-), we obtain a semilattice of rulings $(O \times O, \leq, \wedge, (\top, \top), (\bot, \bot))$ with minimal element $(\bot, \bot)$ and maximal element $(\top, \top)$. Keeping with the definition of a semilattice, for all $r_1 = (o_1^+, o_1^-), r_2 = (o_2^+, o_2^-) \in O \times O$ we have $r_1 \leq r_2 :\iff r_1 \wedge r_2 = r_1 \iff (o_1^+ \wedge o_2^+, o_1^- \wedge o_2^-) = (o_1^+, o_1^-) \iff o_1^+ \leq o_2^+$ and $o_1^- \leq o_2^-$.*

Privacy related regulations often depend on context information; e. g., accessing the data of under age customers for marketing purposes may require the consent of a legal guardian. Hence “age of customer” and “consent of legal guardian” are variables that have to be considered in the evaluation of privacy policies. Restrictions of the type in this example are captured by a 3-valued, many sorted *condition logic*, which is defined over the following *condition vocabulary*.

Definition 4 (Condition Vocabulary). *A* condition vocabulary (S, X, Σ, ρ) *consists of (adapted from [13, Chapter 10]) i) a finite set $S \dot\cup \{l_3\}$ of* sorts *(or* types*) such that S is nonempty; ii) logical connectives $\wedge$, $\vee$ of rank $(l_3, l_3; l_3)$, $\neg$, $\sim$ of rank $(l_3; l_3)$, 0, u, 1 of rank $(\varepsilon; l_3)$; iii) for every sort $s \in S$ the* equality symbol $\doteq_s$ *of rank $(s, s; l_3)$; iv) for every sort $s \in S$ a finite set $X_s = \{x_{s0}, x_{s1}, x_{s2}, \ldots\}$ of* variables*, each variable x_{si} being of rank $(\varepsilon; s)$; we define the family of sets $\mathbf{X} := \{X_s \mid s \in S\}$ and $X := \dot\bigcup_{s \in S} X_s$ the set of all variables; v)* auxiliary symbols *“(” and “)”; vi) a finite $(S \dot\cup \{l_3\})$-ranked alphabet (Σ, ρ) with rank function $\rho : \Sigma \to S^* \times (S \dot\cup \{l_3\})$ of* nonlogical symbols *consisting of:*

- *A finite set $\Sigma_F := \{f \in \Sigma \mid (t; s) := \rho(f) \in S^+ \times S\}$ of* function symbols*. The string t is called* arity *of f and the symbol s* sort *(or* type*) of f.*

- *For each sort $s \in S$ a finite set $\Sigma_C^s := \{c \in \Sigma \mid \rho(f) = (\varepsilon; s)\}$ of* constants. *The family of sets Σ_C^s is denoted by Σ_C.*
- *A finite set $\Sigma_P := \{P \in \Sigma \mid (t; l_3) := \rho(P) \in S^* \times \{l_3\}\}$ of* predicate symbols. *The string t is the* arity *of P, if $t = \varepsilon$, P is a propositional letter.*

We call a condition vocabulary with set of nonlogical symbols Σ a vocabulary Σ.

Having defined a vocabulary for a condition logic, we next define models, the many-sorted algebras, for this logic.

Definition 5 (Many-sorted Algebra). *Given an S-ranked alphabet Σ, a* many-sorted Σ-algebra *M is a pair (M, I) with $M = (M_s)_{s \in S}$ an S-indexed finite family of finite sets $M_s \neq \emptyset$, the* carriers of sort *s, and I an interpretation function $I : \Sigma \to \bigcup_{n \in \mathbb{N}} \bigcup_{s, s_1 \ldots s_n \in S} M_s^{M_{s_1} \times \ldots \times M_{s_n}}$ s. t.: $\forall f \in \Sigma : \rho(f) = (s_1 \ldots s_n, s) \implies I(f) \in M_s^{M_{s_1} \times \ldots \times M_{s_n}}$ and $\forall c \in \Sigma : \rho(c) = (\varepsilon, s) \implies I(c) \in M_s$.*

Now we can define terms, formulas, models and semantic for our condition logic.

Definition 6 (Condition Language). *Let (S, X, Σ, ρ) be a condition vocabulary. The* condition language *$C(S, X, \Sigma, \rho)$ is the set of correctly typed formulas over (S, X, Σ, ρ). Formulas are defined recursively as usual for predicate logic (see [13, Ch. 10]). The* free *variables of a formula $c \in C$ are denoted by* $\mathrm{free}(c)$. *As the condition logic has no quantifiers, these are all variables of c. The semantics of C formulas is defined as usual for the 3-valued Lukasiewicz logic L_3 (cf., e. g., [18, 13, 22]), using the following two definitions.*

Definition 7 (Admissible Model). *A* many-sorted Σ-structure *(M, I) is a many-sorted Σ-algebra, where for all $s \in S$ the symbol for "unknown" $\oslash \in M_s$ and $M_{l_3} = \{0, u, 1\}$. A structure (M, I) is called* admissible *or* admissible model *for a condition language C, provided that $I|_{\Sigma_C} : \Sigma_C \to \bigcup_{s \in S} M_s$ is a surjective map, i. e. there is a constant for every possible data item in the structure M. If a fixed admissible structure is given, we will usually choose one of these constants and use it and the data item interchangibly.*

Definition 8 (Variable Assignment). *An* assignment of sort *s of the variables is a function $\alpha_s \in M_s^{V_s}$. An* assignment *$\alpha = (\alpha_s)_{s \in S}$ is an S-indexed family of assignments of sort s. The set of all assignments for a set of variables X and a structure M is written $\mathfrak{Ass}(X, M)$. An assignment α is* partial *if $\alpha(x) = \oslash$ for some $x \in X$ and* complete *if this is not the case. For the set of complete assignments we write $\mathfrak{Ass}^*(X, M)$. To allow a uniform treatment of policies defined over different vocabularies, we assume a set $\mathcal{X}$ all variables and a set $\mathcal{M}$ all values are taken from. Typically these are sets of strings over a given alphabet, e. g. valid XML expressions; we define the set of assignments $\mathfrak{Ass} := \mathcal{M}^{\mathcal{X}}$.*

2.2 Syntax of NAPS Policies

Having described the basic components of NAPS policies, we now collect them into a vocabulary.

Definition 9 (NAPS Vocabulary). *A (NAPS)* vocabulary V *consists of hierarchies (ordered sets) U, D, P, and A, called user, data, purpose, and action hierarchy, respectively; a condition language C, an admissible structure (M, I) for C and an obligation model O: $V = (U, D, P, A, C(S, X, \Sigma, \rho), (M, I), O)$*

Given two vocabularies V and V' we write $V \subseteq V'$ iff $U \subseteq U'$, $D \subseteq D'$, $P \subseteq P'$, $A \subseteq A'$, $S \subseteq S'$, $\Sigma \subseteq \Sigma'$, $O \subseteq O'$, $\forall s \in S : X_s \subseteq X'_s$, $\forall s \in S : M_s \subseteq M'_s$, $\rho = \rho'|_\Sigma$, $I = I'|_{\Sigma,M}$, where $\forall f \in \Sigma : I'|_{\Sigma,M}(f) := I'(f)|_M$.

As a naming convention, we assume that the components of a vocabulary V are always called as in Definition 9 except if explicitly stated otherwise. In a vocabulary V_i all components also get a subscript i, and similarly for superscripts.

As indicated before, privacy policies make statements about *users* performing an *action* on *data* with a specific *purpose*. Accordingly, a NAPS query is a tuple (u, d, p, a) in the query set $Q(V) := U \times D \times P \times A$ for the given vocabulary V. NAPS queries are not restricted to minimal elements in the hierarchies. This facilitates the handling of policies in scenarios, where a coarse policy, say a company policy referring only to departments, is refined, say to a department policy making statements about workgroups or individuals. In such a scenario elements that are initially minimal may later get children: User John Doe my not be mentioned in the company policy, but may very well appear in a department policy. Also, it may still be of interest to query the company policy with a department, to find out about the (minimum) restrictions for that department.

To be able to treat policies defined on different vocabularies in a uniform manner, we define the semantics for queries outside the given vocabulary. We assume a superset $\mathcal{H}$, in which all hierarchy sets are embedded; in practice it is typically a set of strings or valid XML expressions.

Definition 10 (Query). *For a vocabulary V, we call $Q(V) = U \times D \times P \times A$ the* query vocabulary *associated with V and define an order $\leq$ on $Q(V)$ as follows. For queries $(u, d, p, a), (u', d', p', a') \in Q(V)$ we set $(u, d, p, a) \leq (u', d', p', a') :\iff (u \leq_U u') \wedge (d \leq_D d') \wedge (p \leq_P p') \wedge (a \leq_A a')$. Given a superset $\mathcal{H}$ of the sets U, D, P, A of all considered vocabularies, the set of* all queries *is $Q := \mathcal{H}^4$.*

Whether a rule in a privacy policy applies to a query $q \in Q(V)$ is determined by evaluating a logical expression or guard over the predicate $\leq$. These logical expressions are taken from the query or guard logic.

Definition 11 (Query or Guard Logic). *The* query *or* guard logic *G for a vocabulary V is a boolean predicate logic without quantifiers over i) the* vocabulary *consisting of the binary predicate "$\leq$" and constants $C_G := Q(V)$; ii) the* set of variables *$\{p\}$; iii)* operators $\wedge$, $\vee$, $\neg$, 1, 0*; iv) auxiliary symbols "(", ")".*

We fix a model $M_G := Q(V)$ for G with the interpretation $I_G(\leq) :=\leq$ (on $Q(V)$) and for $q \in C_G$: $I_G(q) := q \in Q(V)$. The semantics are as usual for a boolean predicate logic and we set $g(q) := \mathrm{eval}_{I_G, M_G, \alpha_G : p \mapsto q}\, g$.

Given the definitions above we can now define NAPS policies. Each NAPS policy is defined over a vocabulary V and consists of a ruleset as its central component and a default ruling. The ruleset states how requests are to be treated, while the default ruling provides a safe default behavior for queries that are not (yet) handled by a rule in the ruleset.

Definition 12 (Ruleset and Privacy Policy). *A* ruleset R *over a vocabulary* V *is a subset of* $\mathbb{Z} \times G \times C \times O \times O$. *A rule* $(i, g, c, r) \in R$ *consists of a* priority $i \in \mathbb{Z}$, *a* guard $g \in G$ *(the guard logic), a* condition $c \in C$ *(the condition logic) and a ruling* $r \in O \times O$. *A* privacy *or* NAPS policy $\mathcal{P} = (V, R, dr)$ *is a triple of a vocabulary* V, *a rule-set* R *over* V, *and a* default ruling $dr \in O \times O$. *We call the set of these policies* NAPS *and the subset for a given vocabulary* $\mathrm{NAPS}(V)$.

As a naming convention, we assume that the components of a privacy policy called $\mathcal{P}$ are always called as in Def. 12, and if $\mathcal{P}$ has a sub- or superscript, then so do the components.

2.3 Semantics of NAPS Policies

The *Chief Privacy Officer* (CPO) of a company should be able to define a binding set of base regulations for the departments of a company, that can in turn be refined by the departments. One goal of NAPS is to improve the control over the refinement of rules versus E-P3P, and to facilitate the intuitive handling of partial knowledge, i. e. partial variable assignments α for the condition logic C. Because of that NAPS uses a 3-valued condition logic C. As in E-P3P a condition evaluating to "0" means that a rule does not apply and a condition evaluating to "1" states that the rule applies and terminates evaluation. The third logical value "u" signifies that a rule applies, but evaluation is to proceed. This way we can treat rules that, due to a partial assignment, might (or might not) apply by applying their obligation conjunctively and proceeding with evaluation, yielding a ruling that might be more restrictive than necessary, but never too lenient. The distinction between "u" and "1" can also be used to mark certain rules as "amendable" by lower priority rules or as "final". Hence we speak of *amendable*, *final* and *semi-amendable rules*. Amendable rules can be refined by lower priority rules, while final rules terminate the evaluation of the policy if they apply and can thus not be refined. Semi-amendable rules can be refined in some cases, depending on the variable assignment α for the condition logic C.

Definition 13 (Amendable, Final, Semi-Amendable Rules). *Let* $\mathcal{P} = (V, R, dr)$ *be a policy. Then a rule* $(i, g, c, r) \in R$ *is* amendable *iff* $\forall \alpha \in \mathfrak{Ass}(X, M) : \mathrm{eval}_\alpha(c) \in \{0, u\}$. *Similarly, a rule* $(i, g, c, r) \in R$ *is* final *iff* $\forall \alpha \in \mathfrak{Ass}(X, M) : \mathrm{eval}_\alpha(c) \in \{0, 1\}$. *A rule that is neither amendable nor final is called* semi-amendable.

A rule $(i, g, c, r) \in R$ *is called* strongly amendable *iff* $\exists c' \in C : c = c' \wedge u$, *and a rule* $(i, g, c, r) \in R$ *is called* strongly final *iff* $\exists c' \in C : c = \sim\sim c'$, *with* $=$ *meaning equality up to equivalence transformations of the underlying* L_3 *logic [18, 22].*

Remark 2. Clearly strongly amendable rules are amendable and strongly final rules are final over each vocabulary, over which their symbols are defined.

The terms above describe a rule as such, independent of a specific query $q \in Q(V)$ or assignment $\alpha \in \mathfrak{Ass}(X, M)$. Given a specific query $q \in Q(V)$ and assignment $\alpha \in \mathfrak{Ass}(X, M)$, we may distinguish *applicable*, *terminal* and *non-applicable* rules under q and α. A rule is not applicable under q and α if either the guard or the condition evaluate to 0, if both guard and condition evaluate to 1, we call a rule terminal under q and α, if they evaluate to 1 and u or 1 respectively, the rule is called applicable under q and α. Therefore, while final rules are either terminal under q and α or not applicable, amendable rules are never terminal.

Definition 14 (Applicable and Terminal Rules). *Let a privacy policy $\mathcal{P} = (V, R, dr)$, a query $q \in Q(V)$, and an assignment $\alpha \in \mathfrak{Ass}(X, M)$ be given.*

Then a rule $(i, g, c, r) \in R$ is applicable *iff $g(q) = 1$ and $\mathrm{eval}_\alpha(c) \in \{u, 1\}$; the rule $(i, g, c, r) \in R$ is* terminal *iff $g(q) = 1$ and $\mathrm{eval}_\alpha(c) = 1$.*

By $R_A(\mathcal{P}, q, i, \alpha) := \{(i, g, c, r) \in R \mid g(q) = 1$ and $\mathrm{eval}_\alpha(c) \in \{u, 1\}\}$ we denote the set of applicable rules for priority i. We define the set of terminal rules for priority i as $R_T(\mathcal{P}, q, i, \alpha) := \{(i, g, c, r) \in R \mid g(q) = 1$ and $\mathrm{eval}_\alpha(c) = 1\}$.

Definition 15 (Precedence Range). *For a privacy policy $\mathcal{P} = (V, R, dr)$ and $\mathrm{op} \in \{\max, \min\}$, let $\mathrm{op}(\mathcal{P}) := \mathrm{op}(R) := \mathrm{op}(\{i \mid \exists (i, g, c, r) \in R\})$.*

The semantics of a NAPS policy, i. e. the result of a query given an assignment, is given by Alg. 2.1. A policy is evaluated by simply collecting the obligations of all applicable rules for the given query and assignment conjunctively, descending by priority until a terminal rule is found. Should no rule apply, the default ruling is returned, and should query or assignment be out of vocabulary, an error is returned. In addition to a ruling, the policy evaluation returns a tag $v \in \{\mathsf{f}, \mathsf{a}, \mathsf{d}\}$ stating how policy evaluation terminated: f (final) indicates, that the evaluation terminated with a terminal rule, a (amendable) states, that no terminal rule was found, but a rule was applicable, d indicates that the default rule was applied. These evaluation tags are helpful in defining useful notions of policy refinement, composition and conjunction, purely on the level of evaluations without regard to the internal structure of a policy.

Definition 16 (Semantics). *Let a privacy policy $\mathcal{P} = (V, R, dr)$, $q \in Q$, and $\alpha \in \mathfrak{Ass}$ be given. We define the set of possible* evaluation results *or* evaluations *$E(V) := O \times O \times \{\mathsf{f}, \mathsf{a}, \mathsf{d}\} \subseteq \mathcal{E} := \mathcal{O} \times \mathcal{O} \times \{\mathsf{f}, \mathsf{a}, \mathsf{d}\}$. The* evaluation result *$e = (r, v) := \mathrm{eval}_\alpha(\mathcal{P}, q) \in E(V)$ of policy $\mathcal{P}$ for query q and assignment α is defined by Alg. 2.1, where "`return`" returns its argument and terminates the algorithm. The evaluation tag $v \in \{\mathsf{f}, \mathsf{a}, \mathsf{d}\}$ states if the evalution was terminated by a terminal rule, found non-terminal rules only or if the default rule was applied.*

From the definition of the semantics of a NAPS policy the benefits of a 3-valued condition logic are apparent. The purpose of the logical value u as result of a condition c is as such twofold: i) u may indicate that due to incomplete information (a partial variable assignment with too many $\oslash$'s) one cannot determine if the rule is to be applied. In this case the rule is applied and the obligations are imposed to guarantee correct treatment of data, should it turn out later that the rule was to be applied. Then evaluation proceeds to look for other rules that might also match the incomplete data; ii) u may

Algorithm 2.1: Policy Evaluation

Input: policy $\mathcal{P}$, assignment $\alpha \in \mathfrak{Ass}$, query $q \in \mathcal{Q}$.
Output: NAPS evaluation $e \in E(V)$.

```
if q ∉ Q(V) or α|_X ∉ 𝔄𝔰𝔰(X,M): return e := ((⊥,⊥),f); // invalid input
r := (⊤,⊤);
for i := max(𝒫) downto min(𝒫): // descend by priority i ...
    r := r ∧ ⋀_{(i,g,c,r')∈R_A(𝒫,q,i,α)} r';   // ... picking up the rulings
    if R_T(𝒫,q,i,α) ≠ ∅: return e := (r,f); // ... terminating, if needed
if ∀i ∈ ℤ : R_A(𝒫,q,i,α) = ∅: return e := (dr,d); // no applicable rule
return e := (r,a); // applicable rule(s) but no final rule found
```

indicate that a rule is amendable. An amendable rule has a condition always returning u instead of 1, so that evaluation continues and picks up further rules concerning the query.

A final rule is, if applicable, automatically terminal, i. e. its condition c returns 1. In that case the evaluation stops at the priority level of the rule in question and the obligations accumulated up to and including that level are returned.

Remark 3 (Interdependence of Query and Assignment). In general we expect the variable assignment to be determined by the requested data object. Variables not being fixed by the requested data object are, if not determined globally, set to unknown ("$\oslash$"). A variable "age of customer", e. g., should be set to the age entry in the customer dataset. For corporate customers "age of customer" makes no sense and is hence set to unknown ("$\oslash$"). The same is true if the age of a customer is simply unknown. As a dataset may in principle induce an arbitrary assignment, NAPS views query and assignment as independent and does not model the connection just explained.

2.4 Embedding E-P3P Policies into the NAPS framework

A privacy policy can be regarded as description of a function mapping a query q and an assignment α to an evaluation e. Each such function can be represented by a NAPS policy. To prove this, an algorithm creating the policy in question can be constructed. However, due to the page limit here we omit the proof.

Theorem 1 (Functional Completeness of NAPS). *The set of NAPS policies is functionally complete in the sense that, for an arbitrary but fixed $dr \in O \times O$, we may describe an arbitrary function $f \in (O \times O \times \{\mathsf{a},\mathsf{f}\} \cup (dr, \mathsf{d}))^{Q(V) \times \mathfrak{Ass}(X,M)}$ through a NAPS policy $\mathcal{P}$.*

In particular, every E-P3P policy defined over an obligation model, that can be turned into a meet-semilattice, can be transformed into an equivalent NAPS policy, using the map $(+, o^+) \mapsto (o^+, \perp, \mathsf{f})$, $(-, o^-) \mapsto (\perp, o^-, \mathsf{f})$, $\circ \mapsto (\top, \top, \mathsf{d})$ from E-P3P rulings to NAPS evaluations.

2.5 Refinement and Equivalence of Privacy Policies

It is of interest to determine, if a policy is more specific or restrictive than another. Department policies should be more specific or restrictive than the minimum requirements set in a company policy and it might be important to know if a policy is at least as restrictive than applicable law or a treaty requirement, so that the company fulfills its legal obligations. To be able to state what it means that a policy is more restrictive than another, we first define in which case we consider an evaluation more restrictive or a *refinement* of another.

Definition 17 (Refinement of Evaluations). *Given two evaluations* $e_i = (r_i, v_i) \in E(V_i) \subseteq \mathcal{E}$ *for* $i = 1, 2$*, we say that* e_2 functionally refines e_1*, written* $e_2 \preceq e_1$*, iff* $r_2 \leq r_1$*. We say that* e_2 weakly refines e_1*, written* $e_2 \lesssim e_1$*, iff* $(v_1 = \mathsf{d}$ *and* $v_2 \neq \mathsf{d})$ *or* $(r_2 \leq r_1$ *and* $(v_1 = \mathsf{d}$ *or* $v_2 \neq \mathsf{d}))$*. Finally, defining* $\mathsf{f} \leq \mathsf{a} \leq \mathsf{d}$*, we say that* e_2 refines e_1*,*

$$e_2 \leq e_1 :\iff (v_1 = \mathsf{d} \text{ and } v_2 \neq \mathsf{d}) \text{ or } (v_2 \leq v_1 \text{ and } r_2 \leq r_1). \tag{1}$$

Remark 4 (Refinement of Evaluations). The functional refinement relation $\preceq$ and the weak refinement relation $\lesssim$ on evaluations are reflexive and transitive, as $\leq$ is an order on the rulings. The refinement relation $\leq$ is even an order on the evaluations and we have $e_2 \leq e_1 \implies e_2 \lesssim e_1$.

Note the special treatment of the default ruling in the definitions of weak refinement and refinement. The default ruling is to describe some safe (possibly very restrictive) behavior until an actual rule has been implemented ("stub behavior"). As we expect a refinement to be more specific, refinement may replace the default ruling on some query with a (possibly less restrictive) non-default ruling. Otherwise we demand a refinement to be more restrictive, imposing stronger obligations or turning amendable rulings into final ones.

From evaluations we now extend the notion of refinement to policies. As with evaluations, functional refinement only captures the notion "more restrictive", whereas (weak) refinement states if a policy is more specific than another, taking into account the special semantics of the default ruling and the vocabulary. Since policies are not uniquely determined by their functional behavior, we collect them into equivalence classes and distinguish i) *functionally equivalent* policies, that generate matching rulings for each query and assignment; ii) *equivalent* policies, that have the same vocabulary and generate matching evaluations for each query and assignment; iii) *strongly equivalent* policies, that have the same vocabulary and generate matching evaluations under each vocabulary extension (see Def. 19) for each query and assignment.

Definition 18 (Refinement and Equivalence of Policies). *A policy* $\mathcal{P}'$ *is called* functional refinement *of a policy* $\mathcal{P}$*, written* $\mathcal{P}' \preceq \mathcal{P}$*, iff* $\forall q \in Q, \alpha \in \mathfrak{Ass} : \operatorname{eval}_\alpha(\mathcal{P}', q) \preceq \operatorname{eval}_\alpha(\mathcal{P}, q)$*. A policy* $\mathcal{P}'$ *is called* refinement *of a policy* $\mathcal{P}$*,*

$$\begin{aligned}\mathcal{P}' \leq \mathcal{P} :\iff\ & (\forall q \in Q(V), \alpha \in \{\beta \in \mathfrak{Ass}(X', M') \mid \beta|_X \in \mathfrak{Ass}(X, M)\} : \\ & \operatorname{eval}_\alpha(\mathcal{P}', q) \leq \operatorname{eval}_\alpha(\mathcal{P}, q)) \text{ and } V \subseteq V'.\end{aligned} \tag{2}$$

Policy $\mathcal{P}'$ *is a* weak refinement *of policy* $\mathcal{P}$*, written* $\mathcal{P}' \lesssim \mathcal{P}$*, iff* $V \subseteq V'$ *and* $\forall q \in Q(V), \alpha \in \{\beta \in \mathfrak{Ass}(X', M') \mid \beta|_X \in \mathfrak{Ass}(X, M)\} : \operatorname{eval}_\alpha(\mathcal{P}', q) \lesssim \operatorname{eval}_\alpha(\mathcal{P}, q)$*.*

Finally, two policies $\mathcal{P}$ *and* $\mathcal{P}'$ *are called i)* functionally equivalent*, written* $\mathcal{P}' \approx \mathcal{P}$*, iff* $\mathcal{P}' \preceq \mathcal{P}$ *and* $\mathcal{P} \preceq \mathcal{P}'$*, ii)* equivalent*, written* $\mathcal{P}' \cong \mathcal{P}$*, iff* $\mathcal{P}' \leq \mathcal{P}$ *and* $\mathcal{P} \leq \mathcal{P}'$*,* *iii)* strongly equivalent*, written* $\mathcal{P}' \equiv \mathcal{P}$*, iff* $V = V'$ *and* $\forall W \supseteq V : \mathcal{P}' \uparrow_W \cong \mathcal{P} \uparrow_W$ *(regarding* $\uparrow$ *see Def. 19).*

Remark 5 (Refinement and Equivalence of Policies). The refinement relations $\preceq, \lesssim, \leq$ just defined are reflexive and transitive and we have $\mathcal{P}' \leq \mathcal{P} \implies \mathcal{P}' \lesssim \mathcal{P}$. Moreover, the relations $\approx, \cong, \equiv$ are equivalence relations and we have $\mathcal{P}' = \mathcal{P} \implies \mathcal{P}' \equiv \mathcal{P} \implies \mathcal{P}' \cong \mathcal{P} \implies \mathcal{P}' \approx \mathcal{P}$.

3 NAPS Operators

Having presented the basic definitions of the NAPS framework, we now turn to defining operators. Since the legal requirements and the structures within a company are subject to change we first define a *vocabulary extension* or *up-scoping* operator, that extends the vocabulary over which a policy is defined, e. g. by adding new users, departments, data types or variables.

Definition 19 (Vocabulary Extension, Up-Scoping). *Let* $\mathcal{P} = (V, R, dr)$ *be a privacy policy over the vocabulary* V *and let* V' *be a vocabulary such that* $V \subseteq V'$*. Then* $\mathcal{P} \uparrow_{V'} := (V', R, dr)$ *is the* up-scoping *of* $\mathcal{P}$ *w. r. t.* V'*.*

In a similar fashion we can define a *down-scoping* operator $\downarrow_{V'}$, that restricts a policy $\mathcal{P}$ to a policy $\mathcal{P} \downarrow_{V'}$ over a smaller vocabulary $V' \subseteq V$. The down-scoping operator is useful to extract department relevant data from a company policy or to discard entitites that are no longer of concern.

Due to space limitations we omit a precise definition of the down-scoping operator and refer the interested reader to the full version of this paper [20]. Note however, that down-scoping only makes sense if i) $\leq_{H'} = \leq_H \cap (H' \times H')$ for $H \in \{U, D, P, A\}$; ii) R contains no symbol in $\Sigma \setminus \Sigma'$; and iii) R and dr contain no obligation in $O \setminus O'$. In the sequel, if we make statements about $\mathcal{P} \downarrow_W$ for a vocabulary $W \subset V$, these are to be understood to apply only for vocabularies W, that fulfill the requirements above. In the full paper [20] we also define *ruleset* and *policy reduction* operators "reduce", that remove redundant terms from a ruleset, thereby possibly enlarging the number of vocabularies suitable for down-scoping.

Often scoping leads to a refinement or at least functional refinement of a policy. E. g., extending the vocabulary will generally lead to a refinement (the policy becomes more comprehensive), while restricting it (such that nothing changes on the remaining items) will lead to a functional refinement (the policy will behave the same on the smaller vocabulary, but produce error states otherwise).

Lemma 1 (Refinement Properties of Scoping Operators). *Given a privacy policy* $\mathcal{P} = (V, R, dr)$ *and vocabularies* V', V''*, s. t.* $V' \subseteq V \subseteq V''$*,* $\leq_{H'} = \leq_H \cap (H' \times H')$ *and* $\leq_H = \leq_{H''} \cap (H \times H)$ *for* $H \in \{U, D, P, A\}$*, we have* $\mathcal{P} \uparrow_{V''} \leq \mathcal{P}$*,* $\mathcal{P} \preceq \mathcal{P} \uparrow_{V''}$ *and* $\mathcal{P} \uparrow_{V''} \downarrow_V = \mathcal{P}$*.*

Furthermore, if the symbols in $\Sigma \setminus \Sigma'$ *and in* $X \setminus X'$ *do* not *occur in the ruleset* R*, we have* $\mathcal{P} \downarrow_{V'} \preceq \mathcal{P}$ *and* $\mathcal{P} \leq \mathcal{P} \downarrow_{V'}$*. But in general* $\mathcal{P} \downarrow_{V'} \uparrow_V \not\approx \mathcal{P}$*.*

Analogously as for E-P3P, we define precendence shifts:

Table 1. Ordered Composition

$\parallel$	(r_2, f)	(r_2, a)	(r_2, d)
(r_1, f)	(r_1, f)	(r_1, f)	(r_1, f)
(r_1, a)	$(r_1 \wedge r_2, \mathsf{f})$	$(r_1 \wedge r_2, \mathsf{a})$	(r_1, a)
(r_1, d)	(r_2, f)	(r_2, a)	$(r_1 \wedge r_2, \mathsf{d})$

Table 2. Conjunction

$\wedge$	(r_2, f)	(r_2, a)	(r_2, d)
(r_1, f)	$(r_1 \wedge r_2, \mathsf{f})$	$(r_1 \wedge r_2, \mathsf{a})$	(r_1, a)
(r_1, a)	$(r_1 \wedge r_2, \mathsf{a})$	$(r_1 \wedge r_2, \mathsf{a})$	(r_1, a)
(r_1, d)	(r_2, a)	(r_2, a)	$(r_1 \wedge r_2, \mathsf{d})$

Definition 20 (Precedence Shift). *Let $\mathcal{P} = (V, R, dr)$ be a privacy policy and $j \in \mathbb{Z}$. Then $\mathcal{P} + j := (V, R + j, dr)$ with $R + j := \{(i + j, g, c, r) \mid (i, g, c, r) \in R\}$ is called the* precedence shift *of $\mathcal{P}$ by j. We define $\mathcal{P} - j := \mathcal{P} + (-j)$.*

Remark 6 (Precedence Shift). Clearly $\forall j \in \mathbb{Z} : \mathcal{P} \equiv \mathcal{P} + j$.

3.1 Composition of Policies

A framework for privacy policies should allow to start, say, from a company policy, stating some minimal requirements, and then to add new rules, collected in another policy, to refine the company policy to a department policy. NAPS supports this through the *composition* operator, that refines its first argument by adding the rules of its second argument with lower priority.

Definition 21 (Composition). *Let $\mathcal{P}_1 = (V, R_1, dr_1)$, $\mathcal{P}_2 = (V, R_2', dr_2)$ be privacy policies over a vocabulary V, $R_2' := R_2 - \max(R_2) + \min(R_1) - 1$. Then $\mathcal{P}_1 \parallel \mathcal{P}_2 := (V, R_1 \cup R_2', dr_1 \wedge dr_2)$ is the (ordered)* composition *of $\mathcal{P}_2$ under $\mathcal{P}_1$.*

Composition can also be defined functionally, pointwise on the evaluations of two policies. Differing from E-P3P this is possible, due to the introduction of the evaluation tag $v \in \{\mathsf{f},\mathsf{a},\mathsf{d}\}$. From Def. 21 (Composition) we get

Lemma 2 (Functional Definition of Composition). *For policies $\mathcal{P}_1$, $\mathcal{P}_2$ over the same vocabulary V define the composition $e_1 \parallel e_2$ on evaluations $e_1 = (r_1, v_1), e_2 = (r_2, v_2) \in E(V)$ as in Table 1. Then for all $q \in Q(V)$ and $\alpha \in \mathfrak{Ass}(X, M)$, we have* $\mathrm{eval}_\alpha(\mathcal{P}_1 \parallel \mathcal{P}_2, q) = \mathrm{eval}_\alpha(\mathcal{P}_1, q) \parallel \mathrm{eval}_\alpha(\mathcal{P}_2, q)$.

By construction, the composition of two policies refines the first policy. Namely, from Eq. (2), (1) and Lemma 2 we obtain

Lemma 3. *For policies $\mathcal{P}_1$ and $\mathcal{P}_2$ over the same vocabulary, we have $\mathcal{P}_1 \parallel \mathcal{P}_2 \leq \mathcal{P}_1$.*

3.2 Policy Normalization

Each NAPS policy can be normalized, i. e. transformed into a strongly equivalent policy, that has only strongly final and strongly amendable rules and only one rule per priority level, where all final rules are of lower priority than any amendable rule.

Definition 22 (Normalization). *A ruleset R over the vocabulary V is called* normalized *iff i) $R = R^A \cup R^F$; ii) all rules $(i, g, c, r) \in R^A$ are strongly amendable; iii) all rules $(i, g, c, r) \in R^F$ are strongly final and $r = (\top, \top)$; iv) $\max(R) = 0$; v) $\forall i$ s. t. $\min(R) \leq i \leq 0$ there is exactly one rule of priority i in R denoted by $R(i)$; and vi) $\forall (i, g, c, r) \in R^A \; \forall (i', g', c', r') \in R^F : i' < i$.*
A policy $\mathcal{P} = (V, R, dr)$ is called normalized, *iff the ruleset R is normalized.*

Algorithm 3.1: Policy Conjunction

Input: policies $\mathcal{P}_1 = (V, R_1, dr_1), \mathcal{P}_2 = (V, R_2, dr_2)$
Output: conjunction policy $\mathcal{P} = \mathcal{P}_1 \wedge \mathcal{P}_2$

```
for i = 1,2: if R_i not normalized: R_i := norm(R_i);
dr := dr_1 ∧ dr_2;
R := R_1^A ∪ (R_2^A + min(R_1^A) − max(R_2^A) − 1);  // "union" of amendable parts
i := min(R) − 1;
for j := max(R_1^F) downto min(R_1^F):
    (j, g_1, c_1, r_1) := R_1^F(j);
    for k := max(R_2^F) downto min(R_2^F):
        (k, g_2, c_2, r_2) := R_2(k);
        R := R ∪ {(i, g_1 ∧ g_2, c_1 ∧ c_2, (⊤, ⊤))};  // "pairwise final conjunction"
        i := i − 1;
return P = (V, R, dr)
```

We always write R^A for the subset of strongly amendable rules of a ruleset R, and R^F for the subset of strongly final rules of a ruleset R. If R carries a sub- or a superscript, so does R^A, R^F. Due to the page limit we omit the proof of

Lemma 4 (Ruleset & Policy Normalization). *There is an algorithm that given a ruleset R over the vocabulary V generates a normalized ruleset* $\mathrm{norm}(R)$. *From a policy $\mathcal{P} = (V, R, dr)$ we can generate a normalized policy* $\mathrm{norm}(\mathcal{P}) := (V, \mathrm{norm}(R), dr)$ *such that* $\mathrm{norm}(\mathcal{P}) \equiv \mathcal{P}$.

3.3 Conjunction of Privacy Policies

If two companies A and B cooperate, it may be necessary to apply the privacy policies of A *and* of B with equal priority. Unlike composition, which regards its first argument as being of higher priority—and thus in general does not refine its second argument—conjunction combines two policies in equal right and returns a result that weakly refines both. Specifying the conjunction of two policies such that up-scoping, down-scoping and composition distribute requires careful consideration of the inner workings of the policies involved, as up-scoping may introduce new vocabulary items, on which the effect of the conjunction policy cannot be determined by the functional properties of the original policies alone.

Definition 23 (Policy Conjunction). *Let $\mathcal{P}_1$, $\mathcal{P}_2$ be privacy policies over a vocabulary V. The* conjunction $\mathcal{P} = \mathcal{P}_1 \wedge \mathcal{P}_2$ *of $\mathcal{P}_1$, $\mathcal{P}_2$ is the output of Alg. 3.1.*

Although being defined on policies, policy conjunction exhibits the intuitively desired property that the evaluation of $\mathcal{P}_1 \wedge \mathcal{P}_2$ can be derived pointwise from the evaluations of $\mathcal{P}_1, \mathcal{P}_2$. Due to the page limit we omit the proof of

Lemma 5 (Functional Properties of Policy Conjunction). *For privacy policies $\mathcal{P}_1$, $\mathcal{P}_2$ over the same vocabulary V define the conjunction $\wedge$ on evaluations $e_1 = (r_1, v_1)$, $e_2 = (r_2, v_2) \in E(V)$ as in Table 2. Then for all $q \in Q(V)$ and all $\alpha \in \mathfrak{Ass}(X, M)$ we have* $\mathrm{eval}_\alpha(\mathcal{P}_1 \wedge \mathcal{P}_2, q) = \mathrm{eval}_\alpha(\mathcal{P}_1, q) \wedge \mathrm{eval}_\alpha(\mathcal{P}_2, q)$.

A Note on Disjunction Adding a meaningful disjunction operation to NAPS is possible (without losing closedness). Few use cases impose a disjunctive composition of privacy policies, however. So here we do not discuss policy disjunction.

4 Algebraic Properties of Operators

Now we can prove intuitive algebraic properties, the NAPS operators were designed towards, but due to the page limit we must omit the actual proof of

Theorem 2 (Operator Laws). *Let $\mathcal{P}_1$, $\mathcal{P}_2$ be privacy policies on a vocabulary V and let V', V'' be vocabularies s. t. $V' \subseteq V \subseteq V''$. Then $\mathcal{P}_1 \wedge \mathcal{P}_1 \equiv \mathcal{P}_1$, $\mathcal{P}_1 \wedge \mathcal{P}_2 \equiv \mathcal{P}_2 \wedge \mathcal{P}_1$, $(\mathcal{P}_1 \wedge \mathcal{P}_2) \wedge \mathcal{P}_3 \equiv \mathcal{P}_1 \wedge (\mathcal{P}_2 \wedge \mathcal{P}_3)$, $\mathcal{P}_1 \parallel \mathcal{P}_1 = \mathcal{P}_1$, $(\mathcal{P}_1 \parallel \mathcal{P}_2) \parallel \mathcal{P}_3 = \mathcal{P}_1 \parallel (\mathcal{P}_2 \parallel \mathcal{P}_3)$, $(\mathcal{P}_1 \wedge \mathcal{P}_2) \parallel \mathcal{P}_3 \equiv (\mathcal{P}_1 \parallel \mathcal{P}_3) \wedge (\mathcal{P}_2 \parallel \mathcal{P}_3)$, $(\mathcal{P}_1 \wedge \mathcal{P}_2) \uparrow_{V''} = (\mathcal{P}_1 \uparrow_{V''}) \wedge (\mathcal{P}_2 \uparrow_{V''})$, $(\mathcal{P}_1 \wedge \mathcal{P}_2) \downarrow_{V'} = (\mathcal{P}_1 \downarrow_{V'}) \wedge (\mathcal{P}_2 \downarrow_{V'})$, $(\mathcal{P}_1 \parallel \mathcal{P}_2) \uparrow_{V''} = (\mathcal{P}_1 \uparrow_{V''}) \parallel (\mathcal{P}_2 \uparrow_{V''})$, $(\mathcal{P}_1 \parallel \mathcal{P}_2) \downarrow_{V'} = (\mathcal{P}_1 \downarrow_{V'}) \parallel (\mathcal{P}_2 \downarrow_{V'})$, $\mathcal{P}_1 \wedge \mathcal{P}_2 \lesssim \mathcal{P}_1$, $\mathcal{P}_1 \parallel \mathcal{P}_2 \leq \mathcal{P}_1$, $\mathcal{P}_1 \wedge \mathcal{P}_2 \lesssim \mathcal{P}_1 \parallel \mathcal{P}_2$, $\mathrm{norm}(\mathcal{P}_1) \equiv \mathcal{P}_1$ and $\mathrm{reduce}(\mathcal{P}_1) \equiv \mathcal{P}_1$.*

The next lemma, the proof of which we also omit due to the page limit, shows strong equivalence to have the desired property, that strongly equivalent policies remain strongly equivalent (and thereby also equivalent, i. e. same output under all assignments and queries) under all operators.

Lemma 6 (Properties of Strong Equivalence). *Let $\mathcal{P}_1 \equiv \mathcal{P}'_1$, $\mathcal{P}_2 \equiv \mathcal{P}'_2$ policies over V and let $V' \subset V \subset V''$ then we have $\mathcal{P}_1 \uparrow_{V''} \equiv \mathcal{P}'_1 \uparrow_{V''}$, $\mathcal{P}_1 \downarrow_{V'} \equiv \mathcal{P}'_1 \downarrow_{V'}$, $\mathcal{P}_1 \wedge \mathcal{P}_2 \equiv \mathcal{P}'_1 \wedge \mathcal{P}'_2$ and $\mathcal{P}_1 \parallel \mathcal{P}_2 \equiv \mathcal{P}'_1 \parallel \mathcal{P}'_2$.*

5 Conclusions

NAPS provides a powerful tool to operate on and reason about privacy policies by providing useful operators along with intuitive algebraic relations. Compared to E-P3P, constructing policies seems less involved, as in E-P3P policy evaluation involves a more complicated preprocessing of rulesets ("rule unfolding") [5, 4]. E-P3P rulings can be mapped to NAPS evaluations, opening a possibility to embed E-P3P policies into NAPS. Still NAPS' usefulness remains to be proved: a prototype implementation with an experimental deployment of such a system has to be done to explore practical strengths and weaknesses. We think the already achieved results justify research in this direction.

References

1. P. Ashley et al. E-P3P privacy policies and privacy authorization. In *WPES-02*, pp. 103–109, ACM Press, 2002.
2. M. Backes et al. Efficient Comparison of Enterprise Priv. Policies. In *SAC'04*, pp. 375–382. ACM Press, 2004.
3. M. Backes et al. Unification in Priv. Policy Evaluation – Translating EPAL into Prolog. In *POLICY'04*. IEEE Comp. Soc., 2004.

4. M. Backes et al. An Algebra for Composing Enterprise Priv. Policies. In *ESORICS 2004*, vol. 3193 of *LNCS*, pp. 33–52. Springer, 2004.
5. M. Backes et al. A Toolkit for Managing Enterprise Priv. Policies. In *ESORICS 2003*, vol. 2808 of *LNCS*, pp. 162–180. Springer, 2003.
6. C. Bettini et al. Obligation monitoring in policy management. In *POLICY 2002*, pp. 2–12, 2002.
7. G. Birkhoff. *Lattice Theory*, vol. 25 of *AMS. Colloquium Publications*. AMS, Providence, Rhode Island, 1973.
8. P. A. Bonatti et al. A Component-Based Architecture for Secure Data Publication. In *ACSAC 2001*, pp. 309–318, 2001.
9. P. A. Bonatti et al. A modular approach to composing access control policies. In *CCS-00*, pp. 164–173, ACM Press, 2000.
10. P. A. Bonatti et al. An algebra for composing access control policies. *ACM Trans. on Inf. and Syst. Sec.*, 5(1):1–35, Feb. 2002.
11. S. D. C. di Vimercati and P. Samarati. An authorization model for federated systems. In *ESORICS 1996*, vol. 1146 of LNCS, pp. 99–117. Springer, 1996.
12. Z. Fu et al. IPSec/VPN security policy: Correctness, conflict detection and resolution. In *IEEE Policy 2001*, vol. 1995 of *LNCS*, pp. 39–56. Springer, 2001.
13. J. H. Gallier. *Logic for Comp. Science: Found. of Automatic Theorem Proving*, Ch. 2.5 and 10, pp. 448–456, 483–488. John Wiley & Sons, 1986, `http://www.cis.upenn.edu/~jean/gbooks/logic.html`.
14. V. D. Gligor et al. On the Formal Definition of Separation-of-Duty Policies and their Composition. In *Proc. 19th IEEE Symp. on Sec. & Priv.*, pp. 172–183, 1998.
15. S. Jajodia et al. Provisional authorization. In *Proc. of the E-commerce Sec. and Priv.*, pp. 133–159. Kluwer Academic Publishers, 2001.
16. S. Jajodia et al. Flexible support for multiple access control policies. *ACM Trans. on Database Syst.*, 26(2):214–260, June 2001.
17. G. Karjoth et al. The platform for enterprise privacy practices – privacy-enabled management of customer data. In *CSFW 2002*, vol. 2482 of *LNCS*, pp. 69–84. Springer, 2002.
18. J. Łukasiewicz. Philosophische Bemerkungen zu mehrwertigen Systemen des Aussagenkalküls. *C. R. Soc. Sc. Varsovie*, 23:51–77, 1931.
19. J. D. Moffett and M. S. Sloman. Policy hierarchies for distributed systems management. *IEEE JSAC Special Issue on Network Manag.*, 11(9):1404–1414, 1993.
20. D. Raub and R. Steinwandt. An Algebra for Enterprise Privacy Policies Closed Under Composition and Conjunction. Full Version, 2006, `http://www.crypto.ethz.ch/~raub/publications.html`.
21. C. N. Ribeiro et al. SPL: An access control language for security policies and complex constraints. In *NDSS 2001*, pp. 89–107, Internet Soc., 2001, `http://www.gsd.inesc-id.pt/~avz/pubs/SPL.pdf`.
22. P. H. Schmitt. Nichtklassische Logiken. Script, Universität Karlsruhe, 2004, `http://i12www.ira.uka.de/studium.htm`.
23. R. T. Simon and M. E. Zurko. Separation of Duty in Role-based Environments. In *CSFW 1997*, pp. 183–194, 1997.
24. D. Wijesekera and S. Jajodia. Policy algebras for access control: the propositional case. In *CCS 2001*, pp. 38–47, ACM Press, 2001.
25. D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. *ACM Trans. on Inf. and Syst. Sec.*, 6(2):286–325, 2003.
26. Semilattice. Wikipedia, the free encyclopedia, `http://en.wikipedia.org/wiki/Semilattice`.
27. eXtensible Access Control Markup Language (XACML). OASIS Committee Specification 1.0, Dec. 2002, `http://www.oasis-open.org/committees/xacml`.