A Practical Generic Privacy Language

Moritz Y. Becker, Microsoft Research
Alexander Malkis, IMDEA Software
Laurent Bussard, EMIC

Abstract. We present a declarative language with a formal semantics for specifying both users' privacy preferences and services' privacy policies. Expressiveness and applicability are maximized by keeping the vocabulary and semantics of service behaviours abstract. A privacy-compliant data-handling protocol for a network of communicating principals is described.

1 Introduction

Privacy policy languages allow online services to specify and publish their privacy policies in a machine-readable way. The process of deciding, based on such a policy and the user's privacy preferences, whether or not to disclose user's personal data to the service can thus be automated. But, despite a growing need for privacy-aware technologies [21, 1], adoption of privacy policy languages has been slow. This is due mainly to cultural and economical reasons [6], but existing privacy languages also suffer from *technical* limitations. Above all, due to their limited expressiveness and scope, they cannot express many natural language policies [24]. The problem is that policies are highly heterogeneous, spread out horizontally (coming from a wide variety of application domains with varying vocabulary and requirements) and vertically (expressed across all abstraction layers: legislation, organizational and business requirements, application requirements, low-level access control).

Academic research in this area has focused on developing more expressive privacy languages and logics directly specifying temporal service behaviours [2, 4, 22]. These efforts do not adequately address the problem of limited scope, and are not likely to be widely deployed in the real world for the following reasons.

Firstly, inherently informal interactions still cannot be expressed in these languages (e.g. "[...] we will tell our affiliates to limit their marketing to you [...]", from Citibank's privacy notice). Secondly, it is often unnecessary to precisely specify the meaning of a service behaviour. For instance, it is often sufficient to view "delete data within 7 days" as an atomic entity with some intuitive meaning, without specifying what "delete" or "within" precisely mean and entail. In such cases, precise temporal behaviour specifications are an unnecessary overhead, and force policy authors to think and work at too low a level of abstraction. Thirdly, some amount of ambiguity is often even *desirable* from the point of view of businesses and their legal departments. The precise behaviour semantics of these languages leaves no wiggle room, thus deterring the adoption.

Observing these shortcomings of existing privacy languages, we arrive at the following desirable design goals for a privacy language.

1. A privacy language should be generic in the ontology of service behaviours and hide the semantics of these behaviours by abstraction, in order to support the widest range of policies, both in a horizontal and vertical sense.

2. It should uniformly deal with both sides of disclosure of PII (personally identifiable information), namely user preferences and service policies, and enable satisfaction checking between the two.
3. It should support, and distinguish between, both permissions and obligations over service behaviours, in both user preferences and service policies.
4. As usability, and readability in particular [21], is a critical aspect in any practical policy language, its syntax should be reasonably human-readable.
5. The language built on top of the abstract behaviours should be expressive enough to be widely applicable. In particular, it should support parameterized behaviours, hierarchical data types, recursive relations, and arbitrary constraints.
6. It should support credential-based delegation of authority, which is crucial for modern decentralised and distributed architectures [13].

This paper presents a generic privacy policy language, S4P, designed with these goals in mind. Statements in S4P are meta-statements about abstract parameterised service behaviours. The service behaviours in S4P can be left abstract, which should be sufficient in most cases, or be instantiated to any required level of detail, using any of the many existing specification techniques including temporal logic, obligation languages, transition systems, or even concrete pieces of code. Concrete behaviour ontologies and semantics can be plugged into the language in a modular fashion according to need. The language is also agnostic about how and whether services enforce their policies. This is in line with the implicit trust model which requires users to trust services to adhere to their own policies, and is independent of whether enforcement is established informally via audit trails, by dynamic monitoring, or by static analysis.

Despite its high abstractness, S4P encapsulates notions specific to privacy and data-handling. Apart from language design, we present:

– A proof-theoretic semantics that formalizes which queries are true in a policy or a preference, and, based on this notion, an algorithm to decide when a policy *satisfies* a user's preference (Section 3). This answers the question: "should the user agree to disclose her data?"
– A model-theoretic semantics that formalizes the intuitive meaning of policies and preferences in terms of abstract service behaviours and traces (Section 5). We also show that the satisfaction checking algorithm is sound with respect to the semantics. This answers the question: "what does it mean for a service to comply with its own policy, or with a user's preference?"
– A protocol that regulates communication of user data in a network of users and services (Section 6). This answers the question: "how can S4P enable safe communication in a network of collaborating agents?" The protocol ensures a useful safety property, despite the language's abstractness.

A small case study of a real-world privacy policy is presented in Section 4. Our implementation of S4P is briefly described in Section 7. The paper concludes with a discussion of S4P with regards to the six design goals from above (Section 8). A technical report contains a formalization of the protocol and full proofs [9].

2 Related work

P3P [15] is a language for presenting a website's privacy notice in a structured, machine-readable way. User preferences cannot be expressed in P3P, so ad hoc mechanisms (e.g. the Privacy Tab Slider in Internet Explorer 6 or the syntactic pattern matching language APPEL [16]) for managing preferences and checking them against policies are required. The downside of this approach is that the exact correspondence between preferences and P3P policies is unclear, both syntactically and semantically. Policies can only express what a website *may* do and cannot express positive promises (e.g. "we will notify you if [...]"). Its vocabulary is fixed and web-centric, which limits its expressiveness further [18]. P3P does not satisfy any of the six design goals in Section 1.

DAMP [5] is a formal framework that links an internal privacy policy of an enterprise with its published policy. DAMP's main complexity stems from supporting hierarchical data types using modal operators. S4P supports hierarchical types via constraints (discussed in [7]). Like S4P, DAMP does not fix the vocabulary of actions and data types, and keeps the semantics of actions abstract. As such, it satisfies design goal 1 from Section 1, but not the other goals; for instance, DAMP cannot differentiate between promises and permissions.

Ardagna *et al.* [2] propose a unified language for expressing services' *access control policies*, users' *release policies*, and services' *data-handling policies*. The language does not support first-class obligations that are independent of access control rules [14], and a user's release policy (corresponding to "preference" in our terminology) cannot express requirements on the service's privacy promises. The language commits to a predefined vocabulary and lacks a model semantics.

Barth *et al.* [4] use linear temporal logic to specify positive and negative temporal constraints on the global trace of a network of principals exchanging user data. Satisfaction between preferences and policies is equivalent to checking entailment between two formulas. Hence for data sending actions, their logic satisfies our design goals 2 and 3 (but not the others). Behaviours other than sending data are not supported (particularly, no non-monotonic actions such as deletion), and extensions would be non-trivial as the effects of behaviours on the state are modelled explicitly.

EPAL [3] is a language for specifying and enforcing organizations' internal rules for accessing user data; essentially, it is an access control language (comparable to XACML [23]) with a privacy-centric vocabulary. It does not satisfactorily deal with specifying user preferences and matching them against policies.

3 S4P

Preliminaries. A phrase of syntax is *ground* iff no variables occur in it, and *closed* if no *free* variables (i.e., in the scope of a quantifier) occur in it.

The phrases in S4P are built from a first-order function-less signature Σ with constant symbols **Const** and some set of predicates **Pred**. As usual, an atom a is a predicate symbol applied to an expression tuple of the right arity. The predicate symbols are domain-specific, and we often write atoms in infix notation, e.g. `Alice` is a `NicePerson`.

In order to abstractly represent PII-relevant service behaviours, we assume a further set of predicate symbols **BehSymb**. Atoms constructed from predicates in **BehSymb** are called *behaviour atoms*. These are also usually written in infix notation and may include atoms such as $\langle$delete `Email` within `1 yr`$\rangle$ and $\langle$allow x to control access to `FriendsInfo`$\rangle$.

Further, we assume a domain-specific first-order constraint language whose relation symbols are disjoint from **Pred**, but which shares variables and constants with Σ. A *constraint* is any formula from this constraint language. The only further requirement on the constraint language is the existence of a computable ground validity relation $\models$, i.e., we can test if a ground constraint is true (written $\models c$). The constraint language may, e.g., include arithmetics, regular expressions and constraints that depend on environmental data (e.g. time).

Assertions. An *assertion* α is of the form $\langle E \text{ says } f_0 \text{ if } f_1, \ldots, f_n \text{ where } c\rangle$, where E is a constant from **Const**, the f_i are *facts* (defined below), and c is a *constraint* on variables occurring in the assertion. In an assertion $\alpha = \langle e \text{ says } f \text{ if } f_1, \ldots, f_n \text{ where } c\rangle$, the keyword "if" is omitted when $n = 0$; likewise, "where c" is omitted when $c = \text{true}$.

Henceforth, we keep to the following conventions: x, y denote variables, E, U, S constants from **Const**, e denotes an expression (i.e., either a variable or a constant), c a constraint, a an atom, b a behaviour atom, B a ground behaviour atom, $\mathcal{B}$ a set of ground behaviour atoms, f a fact, F a ground fact, α an assertion, and $\mathcal{A}$ a set of assertions. We use θ for variable substitutions, and γ for ground total variable substitutions (mapping every variable to a constant).

Facts and queries. We can now define the syntax of *facts* f and *queries* q:

$$
\begin{aligned}
f &::= a \mid e \text{ can say } f \mid e \text{ may } b \mid e \text{ will } b \\
q &::= e \text{ says } f? \mid c? \mid \neg q \mid q_1 \wedge q_2 \mid q_1 \vee q_2 \mid \exists x(q)
\end{aligned}
$$

Facts with can say are used to express *delegation of authority* and have a special query evaluation semantics, as defined in the proof system below. Facts involving may and will are not treated specially for query evaluation, but are essential for the privacy-related model semantics in Section 5.

For example, (2)–(18) in Fig. 1 are assertions, and (1) and (19) are queries.

Atomic query evaluation. A query is evaluated in the context of a set of assertions; a closed query evaluates to either true or false. Our query evaluation semantics is a simplified variant of the one from SecPAL [8]. We first define a two-rule proof system that generates ground judgements of the form $\mathcal{A} \vdash E$ says F:

$$
\frac{\begin{array}{c}\langle E \text{ says } f \text{ if } f_1, \ldots, f_n \text{ where } c\rangle \in \mathcal{A} \qquad \models \gamma(c) \\ \text{For all } i \in \{1, \ldots, n\} : \ \mathcal{A} \vdash E \text{ says } \gamma(f_i)\end{array}}{\mathcal{A} \vdash E \text{ says } \gamma(f)}
\qquad
\frac{\begin{array}{c}\mathcal{A} \vdash E_1 \text{ says } E_2 \text{ can say } F \\ \mathcal{A} \vdash E_2 \text{ says } F\end{array}}{\mathcal{A} \vdash E_1 \text{ says } F}
$$

The first rule is derived from the standard modus ponens rule, and the second rule defines delegation of authority using can say.

For example, assertions (2), (3), (4), and (10) in Fig. 1 support the derivation of $\langle$`Alice` says `MS` complies with `COPPA`?$\rangle$: From (3) and (4) we get that `Alice` says that

TRUSTe is a member of COPPASchemes, which with (2) implies that TRUSTe can say who complies with COPPA. Combine it with (10).

Compound query evaluation. The relation $\vdash$ so far only deals with the case where the query is of the basic form $\langle e \text{ says } f?\rangle$. We extend it to all closed queries by interpreting compound queries as formulas in first-order logic. Formally, let $\mathcal{A}$ be a set of assertions and q be a closed query, $\mathcal{M}_{\text{assr}} = \{\alpha \mid \mathcal{A} \vdash \alpha\}$ and $\mathcal{M}_{\text{constr}} = \{c \mid \models c\}$. Then $\mathcal{A} \vdash q$ iff $\mathcal{M}_{\text{assr}} \cup \mathcal{M}_{\text{constr}} \models q$ in first-order logic.

User-service pair. In an *encounter* between a user and a service, the service requests a PII from the user, and the user may agree or disagree to the disclosure. Since the essential parameters of an encounter are the user and the service, it is useful to view these two parameters as a single pair:

A *user-service pair* $\tau = (U, S)$ is a pair of constants denoting the *user* (name) U (the PII owner) and the *service* (name) S (the requester and potential recipient of the PII) during an encounter.

Assertions may contain placeholders $\langle\mathsf{Usr}\rangle$ and $\langle\mathsf{Svc}\rangle$ which get dynamically instantiated during an encounter by U and S, respectively. That way, the same privacy preference can be used for encounters with multiple services, and the same privacy policy can be used for encounters with multiple users.

Will- and may-queries. Two particular classes of queries will serve in defining policy and preference later. In the following, let $\tau = (U, S)$ be a user-service pair.

- A τ-will-*query* q_w is a query in which no subquery of the form $\langle S \text{ says } S \text{ will } b?\rangle$ occurs in the scope of a negation sign ($\neg$).
- A τ-may-*query* q_m is a query in which no subquery of the form $\langle U \text{ says } S \text{ may } b?\rangle$ occurs in a disjunction or in the scope of an existential quantifier or of a negation sign.

The definition above syntactically restricts the queries occurring in a policy or a preference to those that have an intuitive meaning in terms of an upper or a lower bound on behaviours. Disjunction and existential quantification are allowed and have an obvious meaning within a will-query, e.g.

$$\exists t\ (S \text{ says } S \text{ will delete } \texttt{Email} \text{ within } \texttt{t}?\ \wedge\ t \leq \texttt{2yr}?).$$

A may-query, however, represents an upper bound on a service's behaviour, and disjunction does not make much sense in this context. If a service wanted to state that it may possibly use the user's email address for contact *or* for marketing (or possibly not at all), it would specify a *conjunctive* query:

$$U \text{ says } S \text{ may use } \texttt{Email} \text{ for } \texttt{Contact}?\ \wedge\ U \text{ says } S \text{ may use } \texttt{Email} \text{ for } \texttt{Marketing}?$$

If this query is successful in the context of U's preference, the service is permitted to use the email address for contact, marketing, both, or to not use it at all.

Policies and preferences. Now we define the syntax of preferences and policies:

- A τ-*preference* Π_{pr} is a pair $(\mathcal{A}_{pr}, q_w)$ where $\mathcal{A}_{pr}$ is a set of assertions and q_w a closed τ-will-query.

– A τ-*policy* Π_{pl} is a pair $(\mathcal{A}_{pl}, q_m)$ where $\mathcal{A}_{pl}$ is a set of assertions and q_m a closed τ-may-query.

Intuitively, the will-query q_w of the preference specifies a *lower bound* on the behaviours of the service. It expresses *obligations*, i.e., the behaviours that the service must exhibit. The assertions $\mathcal{A}_{pr}$ specify an *upper bound* on the behaviours, i.e., the *permissions*, and typically involve the modal verb may.

The may-query q_m of a policy expresses a *upper bound* on service's behaviours. The query advertises all *possible* relevant behaviours of the service. The service uses q_m to ask for permission for all behaviours that it might possibly exhibit. The assertions $\mathcal{A}_{pl}$ specify a *lower bound* on the behaviours, and typically involve the modal verb will. The service *promises* to exhibit the mentioned behaviours.

This intuition is formalized by a trace semantics in Section 5.

Satisfaction. Should a user agree to the disclosure of her PII? This depends on whether the service's policy *satisfies* her preference. Checking satisfaction consists of two steps. First, every behaviour declared as *possible* in the policy must be *permitted* by the preference. Thus, it is checked that the upper bound specified in the policy is contained in the upper bound specified in the preference. Intuitively, a service must ask for permission upfront for anything that it might do with a user's PII. Second, every behaviour declared as *obligatory* in the preference must be *promised* by the policy. Thus, it is checked that the lower bound specified in the preference is contained in the lower bound specified in the policy. Intuitively, a user asks the service to promise the obligatory behaviours.

Since these dualities are reflected in the language syntax, checking if a service policy satisfies a user preference becomes straightforward in S4P. We just need to check if the may-query in the policy and the will-query in the preference are both satisfied. In general, queries are not satisfied by a single assertion but by a set of assertions. This is because assertions may have conditions that depend on other assertions, and authority over asserted facts may be delegated to other principals. Hence the queries are evaluated against the union of the assertions in the policy *and* the preference.

Definition 1. *A τ-policy $\Pi_{pl} = (\mathcal{A}_{pl}, q_m)$* satisfies *a τ-preference $\Pi_{pr} = (\mathcal{A}_{pr}, q_w)$ iff $\mathcal{A}_{pl} \cup \mathcal{A}_{pr} \vdash q_m \wedge q_w$.*

For example, if $\tau = (\texttt{Alice}, \texttt{MS})$, the τ-policy on the right in Fig. 1 satisfies the τ-preference on the left, because both queries (1) and (19) are derivable from assertions (2)–(18). We will look at this example more closely in the next section.

Complexity. The computational complexity of policy evaluation is usually given in terms of parameterized *data complexity*, where the size of the rules (assertions with conditions) is fixed, and the parameter is the number of facts (assertions without conditions). The data complexity of S4P is polynomial in general and linear for ground policies and preferences; this follows from complexity results on logic programming [20].

(1) $\langle$Svc$\rangle$ says $\langle$Svc$\rangle$ will allow `Alice` to `Edit` ParentalControls?
$\wedge$ `Alice` says $\langle$Svc$\rangle$ complies with `COPPA`?
(2) `Alice` says x can say y complies with `COPPA` if
x is member of `COPPASchemes`.
(3) `Alice` says `FTC` can say
x is member of `COPPASchemes`.
(4) `FTC` says `TRUSTe` is member of `COPPASchemes`.
(5) $\langle$Usr$\rangle$ says $\langle$Svc$\rangle$ may use `Cookies` for x if
$\langle$Svc$\rangle$ will revoke `Cookies` within t
where $t \leq$ `5yr`.
(6) $\langle$Usr$\rangle$ says $\langle$Svc$\rangle$ can say $\langle$Svc$\rangle$ will
revoke `Cookies` within t.
(7) `Alice` says $\langle$Svc$\rangle$ may
allow `Alice` to *action object*.
(8) `Alice` says $\langle$Svc$\rangle$ may revoke `Cookies` within t.
(9) `Alice` says `Alice` is using software
`MSNClient` version 9.5.

(10) `TRUSTe` says `MS` complies with `COPPA`.
(11) `MS` says `MS` will allow $\langle$Usr$\rangle$ to `Edit` ParentalControls if
$\langle$Usr$\rangle$ is member of *msntype*,
msntype supports parental controls,
$\langle$Usr$\rangle$ is using software `MSNClient` version v
where $v \leq 9.5$.
(12) `MS` says `MSNPremium` supports parental controls.
(13) `MS` says `MNSPlus` supports parental controls.
(14) `MS` says `MSN9DialUp` supports parental controls.
(15) `MS` says `MSN` can say x is member of g
where $g \in \{\mathtt{MSN}, \mathtt{MSNPremium}, \mathtt{MSNPlus}, \mathtt{MSN9Dialup}\}$
(16) `MSN` says `Alice` is member of `MSNPremium`.
(17) `MS` says $\langle$Usr$\rangle$ can say $\langle$Usr$\rangle$ is using software
`MSNClient` version v.
(18) `MS` says `MS` will revoke `Cookies` within `2yr`.
(19) $\langle$Usr$\rangle$ says `MS` may use `Cookies` for `AdTracking`? $\wedge$
$\langle$Usr$\rangle$ says `MS` may revoke `Cookies` within `2yr`? $\wedge$
$\langle$Usr$\rangle$ says `MS` may allow $\langle$Usr$\rangle$ to `Edit` ParentalControls?

Fig. 1. Alice's privacy preference (left), Microsoft privacy policy (right)

4 Case Study

Now we discuss an example to illustrate some of the concepts above and S4P's intended usage. In the following, the numbers in parentheses refer to Fig. 1.

Alice's privacy preference. Where does Alice's preference (1–9) come from? There are several possibilities. First of all, she is offered to select among a small number of default preferences for specific application domains. Preferences could be customized using application- or browser-specific user interfaces that do not offer the full expressiveness and flexibility of S4P, but let the user extend or define exceptions to the predefined preferences. User agents can also download default preferences provided by trusted third parties for specific application domains. This case emphasizes the need for a trust delegation mechanism in the language.

Alice cares about online child protection, so her privacy preference contains will-query (1). According to this will-query, Alice requires web services she interacts with to allow her to edit parental control settings. Furthermore, she requires services to comply with the Federal Trade Commission (FTC) Children's Online Privacy Protection Act (COPPA). Of course, Alice does not exactly know which businesses comply with COPPA, so she delegates authority over COPPA compliance to privacy seal programs that certify COPPA compliance, using a "can say" assertion (2). But she does not know the entire list of such programs either, so she delegates authority over such schemes to the FTC (3). She also has a statement from the FTC saying that TRUSTe is such a scheme (4).

Alice's may-assertions allow any service to use cookies for any purpose as long as the service promises that the cookies expire within five years (5,6). Assertions (7,8) are default statements allowing service behaviours that Alice is asking for.

In our scenario, Alice uses MSN Client to access content from MSN, and has an assertion (9) stating the version of the client software (she may also have additional assertions stating other environment variables).

Microsoft's privacy policy. The English statements in *italics* are taken verbatim from Microsoft's Online Privacy Statement[1].

Microsoft is a member of the TRUSTe Privacy Program. This means that Microsoft complies with a number of privacy standards including, in particular, COPPA (10). *If you have an MSN Premium, MSN Plus, or MSN 9 Dial-Up account, and use MSN Client software version 9.5 or below, you can choose to set up MSN Parental Controls for the other users of that account* (11–14). The various types of MSN membership are delegated to MSN, using can say (15).

MSN knows that Alice has a MSNPremium account (16). In our implementation, such assertions can be created on the fly during evaluation using interfaces to databases and directory services such as SQL Server and Active Directory.

Microsoft believes a user's claim about the version of her client (17).

When we display online advertisements to you, we will place a [sic] *one or more persistent cookies on your computer in order to recognize your computer each time we display an ad to you* (19). *The cookies we use for advertising have an expiry date of no more than 2 years* (18).

The may-query (19) explicitly mentions all behaviours for this encounter.

Satisfaction evaluation. Does the policy satisfy Alice's preference? Satisfaction is checked by evaluating Alice's will-query and the service's may-query against the union of the assertions in both preference and policy. The will-query (1) first checks whether the service allows Alice to edit parental control settings. The answer is yes according to assertion (11), because Alice is a member of MSN Premium according to MSN (16) which has been delegated authority over MSN Premium memberships (15). Furthermore, MSN Premium accounts support parental controls according to (12), and Alice is using a version of MSN client that supports parental controls (9) and is trusted on that fact (17).

The second part of (1) checks compliance with COPPA. This is established via a delegation from Alice to TRUSTe using (2) and (10). The condition in (2) is satisfied by another delegation chain, from Alice to FTC, using (3) and (4).

The may-query (19) consists of three conjuncts. The first one is satisfied by Alice's assertion (5) which in turn depends on (6) and Microsoft's will-assertion (18). The remaining two conjuncts are satisfied by Alice's may-assertions (7,8).

Hence Alice's preference is satisfied by the policy, so her user agent is willing to disclose her PII to the website.

5 Trace Semantics

Def. 1 induces an algorithm, based on query evaluation, for checking if a policy satisfies a preference, but it does not show that the algorithm is *correct*. As yet, no definition of "correct" exists. This section formalizes a notion of correctness and proves correctness of the satisfaction checking procedure.

Behaviour function and traces. Policies and preferences bound services' behaviours. We are interested in whether a particular run, or *trace*, of a service *complies* with a

[1] Retrieved from http://privacy.microsoft.com/en-gb/fullnotice.mspx on 16/09/2010.

policy or a preference. Since we care only about PII-relevant behaviours exhibited by a trace, we keep the notion of trace as abstract as possible. We assume a set whose elements are called *traces*, as well as an *abstract behaviour function* **Beh** which maps each trace to a set of ground behaviour atoms. In order to maximize generality of our language, we make no further assumptions on **Beh**. Intuitively, a trace t exhibits exactly the behaviours in $\mathbf{Beh}(t)$. (Conversely, a ground behaviour atom can be seen as a trace property.)

Definition 2. *A trace t* complies *with a set of traces T iff $t \in T$. A set of traces T_1 is* at least as strict *as a set of traces T_2 iff $T_1 \subseteq T_2$.*

5.1 Trace Semantics of Policies

To specify the trace semantics of a policy, we need two auxiliary relations.

Promised obligations. Let $\tau = (U,S)$, let $\mathcal{A}$, $\mathcal{A}_{pl}$ be sets of assertions, and $\mathcal{B}$ a set of ground behaviour atoms. The relation $\mathcal{B} \models^{\mathrm{wa}}_{\tau,\mathcal{A}} \mathcal{A}_{pl}$ holds if the behaviours in $\mathcal{B}$ include all behaviours promised by will-assertions in $\mathcal{A}_{pl}$ in the context of foreign assertions $\mathcal{A}$ (later, $\mathcal{A}$ will come from the user preference):

$$\mathcal{B} \models^{\mathrm{wa}}_{\tau,\mathcal{A}} \mathcal{A}_{pl} \quad \text{iff} \quad \mathcal{B} \supseteq \{B \mid \mathcal{A} \cup \mathcal{A}_{pl} \vdash S \text{ says } S \text{ will } B\}\,.$$

Queried permissions. Let $\tau = (U,S)$, $\mathcal{A}$ be a set of assertions, $\mathcal{B}$ a set of ground behaviour atoms, and q_m a τ-may-query. The relation $\mathcal{B} \models^{\mathrm{mq}}_{\tau,\mathcal{A}} q_m$ holds if all behaviours in $\mathcal{B}$ are contained in the behaviours that *may* be exhibited, as specified by q_m, in the context of $\mathcal{A}$ (later, $\mathcal{A}$ will come from both the policy and the preference). The relation is defined as the smallest relation satisfying:

$$\mathcal{B} \models^{\mathrm{mq}}_{\tau,\mathcal{A}} U \text{ says } S \text{ may } B?\,,\ \text{if } \mathcal{B} \subseteq \{B\}\,;$$
$$\mathcal{B} \models^{\mathrm{mq}}_{\tau,\mathcal{A}} q_1 \wedge q_2\,,\ \text{if } \exists\, \mathcal{B}_1, \mathcal{B}_2 \text{ such that } \mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2,\ \mathcal{B}_1 \models^{\mathrm{mq}}_{\tau,\mathcal{A}} q_1 \text{ and } \mathcal{B}_2 \models^{\mathrm{mq}}_{\tau,\mathcal{A}} q_2\,;$$
$$\emptyset \models^{\mathrm{mq}}_{\tau,\mathcal{A}} q\,,\ \text{if } \mathcal{A} \vdash q \text{ and no subquery of the form } \langle U \text{ says } S \text{ may } B?\rangle \text{ occurs in } q\,.$$

Trace semantics of a policy. The following definition formalizes the intuitive meaning of a policy: a policy characterizes all those traces that respect both the lower and upper bounds on behaviours (as expressed by the will-assertions and the may-query, respectively, in the context of an additional set of assertions $\mathcal{A}$).

Definition 3. *Let $\tau = (U,S)$, $\Pi_{pl} = (\mathcal{A}_{pl}, q_m)$ be a τ-policy, and $\mathcal{A}$ a set of assertions. Then $[\![\Pi_{pl}]\!]^{pl}_{\tau,\mathcal{A}}$ denotes the set of all traces t such that*

$$\mathbf{Beh}(t) \models^{wa}_{\tau,\mathcal{A}} \mathcal{A}_{pl} \quad \textit{and} \quad \mathbf{Beh}(t) \models^{mq}_{\tau,\mathcal{A}_{pl}\cup\mathcal{A}} q_m.$$

Example. Let $\tau = (\texttt{Alice}, \texttt{MS})$ and Π_{pl} consists of (10–19) from Fig. 1. Let $B_1 = \langle \text{allow } \texttt{Alice} \text{ to } \texttt{Edit} \text{ ParentalControls}\rangle$, $B_2 = \langle \text{revoke } \texttt{Cookies} \text{ within } \texttt{2yr}\rangle$, and $B_3 = \langle \text{use } \texttt{Cookies} \text{ for } \texttt{AdTracking}\rangle$. Let $\mathcal{A}$ consist of (2–9). Then $[\![\Pi_{pl}]\!]^{\mathrm{pl}}_{\tau,\mathcal{A}}$ denotes the set of all traces t such that

$$\{B_1, B_2\} \subseteq \mathbf{Beh}(t) \subseteq \{B_1, B_2, B_3\}\,,$$

which corresponds with the intention of the privacy policy described in Section 4.

5.2 Trace Semantics of Preferences

We specify the trace semantics of a preference by two other auxiliary relations.

Permissions. Let $\tau = (U,S)$, let $\mathcal{A}$, $\mathcal{A}_{pr}$ be sets of assertions, and $\mathcal{B}$ a set of ground behaviour atoms. The relation $\mathcal{B} \models^{\mathrm{ma}}_{\tau,\mathcal{A}} \mathcal{A}_{pr}$ holds if all behaviours in $\mathcal{B}$ are contained in the set of behaviours permitted by the may-assertions in $\mathcal{A}_{pr}$ in the context of foreign assertions $\mathcal{A}$ (later, $\mathcal{A}$ will come from the service policy):

$$\mathcal{B} \models^{\mathrm{ma}}_{\tau,\mathcal{A}} \mathcal{A}_{pr} \quad \text{iff} \quad \mathcal{B} \subseteq \{B \mid \mathcal{A} \cup \mathcal{A}_{pr} \vdash U \textsf{ says } S \textsf{ may } B\}\,.$$

Obligations. Let $\tau = (U,S)$, $\mathcal{A}$ be a set of assertions, $\mathcal{B}$ a set of ground behaviour atoms, and q_w a τ-will-query. The relation $\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_w$ holds if the behaviours in $\mathcal{B}$ include all behaviours specified as required by q_w, in the context of $\mathcal{A}$ (later, $\mathcal{A}$ will come from both the service policy and the user preference). The relation is defined as the smallest relation satisfying the following:

$$\begin{aligned}
&\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} S \textsf{ says } S \textsf{ will } B?\,, \quad \text{if } \mathcal{B} \supseteq \{B\}\,;\\
&\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_1 \wedge q_2\,, \quad \text{if } \mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_1 \text{ and } \mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_2\,;\\
&\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_1 \vee q_2\,, \quad \text{if } \mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_1 \text{ or } \mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_2\,;\\
&\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} \exists x(q)\,, \quad \text{if there is } E \in \mathbf{Const} \text{ such that } \mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q[E/x]\,;\\
&\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q\,, \quad \text{if } \mathcal{A} \vdash q \text{ and no subquery of the form } \langle S \textsf{ says } S \textsf{ will } B?\rangle \text{ occurs in } q\,.
\end{aligned}$$

Trace semantics of preferences. The following definition formalizes the trace semantics of a preference in the context of a set of assertions.

Definition 4. *For a user-service pair $\tau = (U,S)$, a τ-preference $\Pi_{pr} = (\mathcal{A}_{pr}, q_w)$, and a set $\mathcal{A}$ of assertions, $[\![\Pi_{pr}]\!]^{pr}_{\tau,\mathcal{A}}$ is the set of all traces t for which*

$$\mathbf{Beh}(t) \models^{ma}_{\tau,\mathcal{A}} \mathcal{A}_{pr} \quad \textit{and} \quad \mathbf{Beh}(t) \models^{wq}_{\tau,\mathcal{A}_{pr}\cup\mathcal{A}} q_w\,.$$

Example. Let $\tau = (\texttt{Alice}, \texttt{MS})$ and Π_{pr} consists of (1–9) from Fig. 1. Let $\mathcal{A}$ consist of (10–18), $B_1 = \langle \textsf{allow } \texttt{Alice} \textsf{ to } \texttt{Edit} \textsf{ ParentalControls}\rangle$, and

$$\mathcal{B} = \{\textsf{allow } \texttt{Alice} \textsf{ to } x\, y, \textsf{revoke } \texttt{Cookies} \textsf{ within } x, \textsf{use } \texttt{Cookies} \textsf{ for } x \mid x, y \in \mathbf{Const}\}.$$

Then $[\![\Pi_{pr}]\!]^{\mathrm{pr}}_{\tau,\mathcal{A}}$ denotes the set of all traces t such that

$$\{B_1\} \subseteq \mathbf{Beh}(t) \subseteq \mathcal{B},$$

which corresponds with the intention of Alice's preference from Section 4.

5.3 Satisfaction and Compliance

Now we link up proof-theoretic satisfaction with model-theoretic compliance. Assuming that a service trace complies with the service's own policy, the theorem tells us that successfully evaluating all queries is indeed sufficient for guaranteeing that the service's trace also complies with the preference.

Theorem 1. *Let $\Pi_{pl} = (\mathcal{A}_{pl}, q_m)$ be a τ-policy and $\Pi_{pr} = (\mathcal{A}_{pr}, q_w)$ a τ-preference. If a trace t complies with $[\![\Pi_{pl}]\!]^{pl}_{\tau,\mathcal{A}_{pr}}$ and Π_{pl} satisfies Π_{pr}, then t complies with $[\![\Pi_{pr}]\!]^{pr}_{\tau,\mathcal{A}_{pl}}$.*

This theorem is completely independent of any concrete instantiation of traces, of the behaviours, and of the **Beh** mapping. The essential correctness property for $S4P$ holds *despite* its abstractness. (Of course, if behaviour-specific properties are to be proved, then **Beh** needs to be filled with some structure.)

6 Safe data handling

In this section we describe a protocol for PII disclosure in a network of users and services that use S4P to express their preferences and policies, respectively. The protocol also regulates transitive communication of PIIs to third parties and evolution of privacy policies. The protocol guarantees privacy of users' PIIs.

User-service encounter. If a service S wishes to collect a PII from a user U, then the following steps are performed (here, $\tau = (U, S)$):

1. U and S decide on a τ-preference Π_{pr} and a τ-policy Π_{pl}, respectively, to be used for this encounter. These may be fixed or result from negotiation.
2. If Π_{pl} satisfies Π_{pr}, then U sends PII to S, otherwise the protocol is aborted. The trust model dictates who checks satisfaction: U (as the main stakeholder), S (wishing to keep parts of its policy secret), or a trusted third party. Availability of computational resources may also influence the decision.
3. S keeps a copy of Π_{pl} and Π_{pr} together with the PII.

Transitive service-service encounter. In most scenarios, disclosing a PII P to a third party S' represents a privacy-relevant behaviour, which should be denoted by a behavioural atom $\langle \mathtt{send}\ P\ \mathtt{to}\ S' \rangle$ (e.g. $\langle \mathtt{send\ Email\ to\ eMarketing} \rangle$) which the **Beh** mapping should keep track of.

A service S may thus only disclose a PII to a third party S' if

1. The policy of S allows the disclosure, and
2. The policy of S' complies with U's preference. Again, the trust model dictates the place to check satisfaction, e.g. at S (not requiring to trust S' on checking), at S' (who might have more resources), or at a trusted third party.

Policy evolution. A service may wish to alter its policy even after having collected the PII. For example, a service may want to disclose the PII to a previously unknown third party, even though the behaviour corresponding to the disclosure action was not declared in the may-assertions in the service's policy. Or it may wish *not* to delete PII despite having promised it in the will-query.

Strictly speaking, both cases represent compliance violations of the service's own original policy. Sometimes such violations should be permitted as long as the new behaviours still comply with the user's original preference. In this scheme, the service would need to alter its policy in such a way that the new behaviours comply with the new policy. It then has to check if the new policy still satisfies the preference. If so, the service may start complying with the new policy, otherwise it must continue complying with the original policy. This scheme guarantees that the service still complies with the user's preference.

Privacy guarantee. Assuming users and services follow the protocol and that all services comply with their own policies, the following safety property holds.

- If a service S possesses U's PII P, either U has sent P earlier to S directly,
- or else S obtained P via a third-party exchange from some service $\tilde{S}$ which possessed P at that time, and the user's preference says that $\tilde{S}$ may send P to S.
- In either case, the trace of S complies with the user's preference.

A formalization of the protocol and of the safety property is found in [9].

7 Implementation

Our prototype implementation focuses on three phases: evaluating policies and preferences, enforcing policies (including disclosure), and verifying trace compliance.

Evaluating policies and preferences. During an encounter, the service discloses its interface, i.e., the type of the required PII, and the associated privacy policy. The privacy policy is evaluated against the privacy preference as described in Section 3. When one or more PII have the required type and a suitable preference, the user is given a choice in a privacy-aware identity selection protocol. If the satisfaction check fails, the user can stop or modify her preferences.

We found that for typical policies, our implementation of satisfaction checking completes within a few milliseconds, even in the context of 10^6 atomic assertions.

Enforcing policies. Services store collected PIIs and keep track of associated rights and obligations by attaching the correspondent "sticky" preference. Obligations are enforced by reacting to external and scheduled events. Before an action is performed on a collected PII, queries are evaluated against the attached preference. Services record privacy-relevant behaviour in execution traces.

Verifying compliance of traces. Execution traces can be used by internal or external auditors in order to check the behaviour of services. Traces are verified according to the trace semantics given in Section 5.

Our implementation of S4P is based on the SecPAL [8] evaluation engine implementation, extended with generic predicates and the may/will-constructs. The evaluation process begins by translating each assertion into constrained Datalog clauses. Queries against the resulting constrained Datalog program are evaluated using a resolution algorithm with tabling [17] in order to guarantee termination even with recursive facts in policies and preferences. The translation preserves S4P's query semantics: a query is true in the context of S4P's assertions iff the corresponding Datalog query evaluates to true against the Datalog program.

A successful query can be visualized by a proof viewer that graphically displays the deduction steps in a proof graph; a failed query can be analysed using our logical abduction tool [10]. In future work, we plan to adapt the tool to suggest modifications of privacy preferences in the case of non-satisfaction.

8 Evaluating S4P's design

This section briefly discusses S4P's language design with regards to the six design goals listed in Section 1.

Generality and abstractness. Abstractness avoids premature commitment to a limited set of features suitable for one particular application domain, but not necessarily for another. It allows concrete ontologies and semantic specifications to be plugged in flexibly, depending on the context and needs. Abstractness is thus conducive to a modular language design, simplifying formal reasoning. As we have showed in this paper, useful correctness properties can be established with relatively little effort, without having to instantiate the temporal and stateful semantics of behaviours.

S4P is abstract in several aspects. First, the vocabulary is kept abstract. Even though most websites' natural language privacy statements have a common structure (e.g. adhering to the Safe Harbor Privacy Principles), with details on notification, user choice, third party disclosure, user access, and security, their vocabularies vary greatly, especially across different application domains.

Second, we have kept the semantics of behaviours abstract by assuming a mapping from traces to behaviour atoms. In most cases it is sufficient to agree on the semantics of behaviours only informally, especially for behaviours involving human interaction. Our framework facilitates such partial informality by providing the abstract level of behaviour atoms. If a more formal treatment is needed, our framework can be used to concretize the meaning of behaviours to any desired level. Complex privacy obligations [22] and temporal logic to express trace constraints [4] are examples of how our abstract notion of behaviour could be concretized.

Third, we are not tied to a specific compliance enforcement model. In practice, automatically enforcing compliance is unfeasible or unnecessary; instead, informal methods such as auditing are used. To automate enforcement, the most direct way is to implement a reference monitor for dynamically checking the permissions, accompanied by an obligation monitoring system [12, 19]. For simple systems, it may be possible to enforce compliance by static analysis, as has been done for cryptographic protocols and access control policies [11].

Uniform treatment of preferences and policies. In S4P, both preferences and policies are uniformly expressed as assertions and queries in a single language. Satisfaction checking between policies and preferences reduces to simple query evaluation.

Support for both permissions and obligations. S4P introduces two modal verbs for specifying upper bounds (may) and lower bounds (will) on service behaviours. This minimal syntactic construct is sufficient for expressing permissions, promises, and obligations, as formalized in Section 5.

Human-readable syntax. The case study from Section 4 showed that real-world online policy statements in natural language can be translated into S4P fairly directly in a way that preserves human readability to a reasonable degree. This is achieved by S4P's infix notation for phrases and the restriction of assertions to essentially the Horn logic fragment, which can be written as if-clauses.

Expressiveness. S4P's relatively high expressiveness compared to other privacy languages is mainly due to its abstractness, but also to a number of language features. First, the application-specific predicates are parameterized, which allows the modelling of arbitrary relations. Second, the if-conditions of assertions are recursive, which is necessary for transitive trust relations. And third, the where-clause may contain arbitrary application-specific constraints, including arithmetic and string ones, and functions for retrieving environmental data.

Support for delegation. The need for trust policies has been long recognized in authorization logics, which has led to the development of language construct for delegation of authority. But trust and delegation is equally important in privacy policies (see e.g. Section 4). S4P supports delegation by qualifying all statements with the says-modality and providing the can say primitive to allow utterances to be dependent on other principals' utterances.

Conclusion. Summarizing, we believe that the abstractness of S4P, in conjunction with the other design goals from Section 1, makes it a particularly attractive privacy language in terms of expressiveness, applicability, usability, and for formal analysis.

References

1. A. Antón, J. Earp, D. Bolchini, Q. He, C. Jensen, and W. Stufflebeam. The lack of clarity in financial privacy policies and the need for standardization. *IEEE Security & Privacy*, 2(2):36–45, 2004.
2. C. A. Ardagna, M. Cremonini, S. D. C. di Vimercati, and P. Samarati. A privacy-aware access control system. *Journal of Computer Security*, 16(4):369–397, 2008.
3. P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise Privacy Authorization Language (EPAL 1.2). Technical report, IBM, Nov. 2003.
4. A. Barth, A. Datta, J. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *IEEE Symposium on Security and Privacy*, May 2006.
5. A. Barth and J. Mitchell. Enterprise privacy promises and enforcement. In *Proceedings of the 2005 Workshop on Issues in the Theory of Security*, pages 58–66. ACM, 2005.
6. P. Beatty, I. Reay, S. Dick, and J. Miller. P3P adoption on e-Commerce web sites: a survey and analysis. *IEEE Internet Computing*, pages 65–71, 2007.
7. M. Y. Becker. SecPAL formalisation and extensions. Technical Report MSR-TR-2009-127, Microsoft Research, 2009.
8. M. Y. Becker, C. Fournet, and A. D. Gordon. Design and semantics of a decentralized authorization language. In *IEEE Computer Security Foundations Symposium*, 2007.
9. M. Y. Becker, A. Malkis, and L. Bussard. S4P: A Generic Language for Specifying Privacy Preferences and Policies. Technical Report MSR-TR-2010-32, Microsoft Research, 2010.
10. M. Y. Becker and S. Nanz. The role of abduction in declarative authorization policies. In *Symposium on Practical Aspects of Declarative Languages (PADL)*, 2008.
11. J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement types for secure implementations. In *Computer Security Foundations Symposium*, 2008.
12. C. Bettini, S. Jajodia, X. Wang, and D. Wijesekera. Obligation monitoring in policy management. In *Policies for Distributed Systems and Networks*, 2002.

13. M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.
14. M. Casassa Mont and F. Beato. On parametric obligation policies: Enabling privacy-aware information lifecycle management in enterprises. In *IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 51–55, 2007.
15. L. Cranor, B. Dobbs, S. Egelman, G. Hogben, J. Humphrey, M. Langheinrich, M. Marchiori, M. Presler-Marshall, J. Reagle, M. Schunter, D. A. Stampley, and R. Wenning. *The Platform for Privacy Preferences 1.1 (P3P1.1) Specification*. W3C, Nov. 2006.
16. L. Cranor, M. Langheinrich, and M. Marchiori. *A P3P Preference Exchange Language 1.0*. W3C, Apr. 2002. `http://www.w3.org/TR/P3P-preferences`.
17. S. W. Dietrich. Extension tables: Memo relations in logic programming. In *Symposium on Logic Programming*, pages 264–272, 1987.
18. H. Hochheiser. The platform for privacy preference as a social protocol: An examination within the U.S. policy context. *ACM Transactions on Internet Technology*, 2(4), 2002.
19. K. Irwin, T. Yu, and W. H. Winsborough. On the modeling and analysis of obligations. In *Computer and communications security*, 2006.
20. A. Itai and J. A. Makowsky. Unification as a complexity measure for logic programming. *Journal of Logic Programming*, 4(2), 1987.
21. C. Jensen and C. Potts. Privacy policies as decision-making tools: an evaluation of online privacy notices. In *Human factors in computing systems*, 2004.
22. Q. Ni, E. Bertino, and J. Lobo. An obligation model bridging access control policies and privacy policies. In *Access control models and technologies*, 2008.
23. OASIS. *eXtensible Access Control Markup Language (XACML) Version 2.0 core specification*, 2005. At `www.oasis-open.org/committees/xacml/`.
24. W. H. Stufflebeam, A. I. Antón, Q. He, and N. Jain. Specifying privacy policies with P3P and EPAL: lessons learned. In *Workshop on privacy in the electronic society*, 2004.