S4P: A Generic Language for Specifying Privacy Preferences and Policies

Moritz Y. Becker
Microsoft Research, Cambridge, UK

Alexander Malkis
IMDEA Software, Madrid, Spain

Laurent Bussard
European Microsoft Innovation Centre, Aachen, Germany

April 2010, updated August 2010

Technical Report
MSR–TR–2010–32

Microsoft Research
Roger Needham Building
7 J.J. Thomson Avenue
Cambridge, CB3 0FB
United Kingdom

S4P: A Generic Language for Specifying Privacy Preferences and Policies

Moritz Y. Becker
Microsoft Research, Cambridge, UK

Alexander Malkis
IMDEA Software, Madrid, Spain

Laurent Bussard
European Microsoft Innovation Centre, Aachen, Germany

April 2010, updated August 2010

Abstract

This paper presents S4P, a declarative language for specifying both users' privacy preferences and services' privacy policies. Preferences and policies are uniformly expressed as assertions and queries written in SecPAL extended with two modal verbs, may and will, and can express both permissions and obligations. Checking if a user's preference is satisfied by a service's policy is simple as it only involves evaluating the queries against the assertions. Expressiveness and applicability are maximized by keeping the vocabulary and semantics of service behaviours abstract. The language's model-theoretic semantics is given in terms of abstract service traces, and formalizes the notion of service compliance with respect to a policy or a preference.

Contents

1 Introduction

Businesses have strong economic incentives to aggregate customers' personal data over the Internet. However, a US study conducted by the Progress & Freedom Foundation (PFF) in 2001 reported a surprising finding: within the course of two years, the amount of personal information collected by commercial websites had actually dropped significantly [1]. What incentives could there be for businesses to collect *less* data?

First of all, the late 1990's saw increased pressure from regulators with regard to privacy and data protection. The European Union's privacy directive 95/46/EC was transposed into national laws in 1998; in the US, the Health Insurance Portability and Accountability Act (HIPAA, enacted 1996) regulates privacy in the health sector, the Gramm-Leach-Bliley Act (enacted 1999) in the financial sector, and the Children's Online Privacy Protection Act (COPPA, enacted 1998) regarding children's online data. Indeed, many companies have faced legal conflicts over online privacy issues (recent examples include Facebook [17, 31], Google Books [32] and Sony [19]).

Moreover, consumers' increased concern over online privacy has had an effect on which companies users are willing to do business with: a study by Forrester Research concluded that $15 billion in e-commerce revenue was lost in 2001 due to consumers' privacy worries [25]. Publicly claiming to collect less personal information may thus be seen as a competitive advantage.

Indeed, the PFF study also found that by 2001, 99% of the most popular dot-com websites were posting a privacy policy, a public document describing a site's data-handling practices and privacy promises [1]. The intention of such a policy is to inform users and let them use it as a basis for deciding whether or not to disclose their data. Nevertheless, despite their concern over privacy, the vast majority of users do not bother reading privacy policies. In any case, most users would likely not be able to fully comprehend the policies anyway. Jensen and Potts [24] conclude in their usability study that privacy policies written in natural language are "essentially unusable as decision-making aids for a user concerned about privacy". Moreover, natural language policies are often ambiguous and inconsistent [2].

Privacy policy languages such as P3P [14] 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 personal user data to the service can thus be automated. But despite the growing apparent need for such technologies, adoption of P3P has been slow. This is due mainly to cultural and economical reasons [7] that apply to other privacy enhancing technologies as well, but existing privacy languages also suffer from technical limitations. Above all, due to their limited expressiveness and scope, they cannot express many statements commonly found in natural language policies. Stufflebeam *et al.* [30] point out high-level organization-wide obligations in particular and cite examples from a health insurance's natural language privacy notice (e.g. "[. . .] locations that maintain confidential information have procedures for accessing, labeling and storing confidential records.") that cannot be expressed in P3P and similar existing languages.

More generally, the problem is that policies are highly heterogeneous; they are spread out both 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 focussed on developing more expressive, feature-rich privacy languages, or logics that can directly encode temporal and stateful service behaviours [3, 27, 5, 28]. Although many of these languages are indeed more expressive than P3P, we believe that 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, many high-level policies such as organization-wide ones or ones involving human interaction still cannot be expressed in these languages, as they are inherently informal (e.g. "[. . .] we will tell our affiliates to limit their marketing to you [. . .]", from Citibank's privacy notice). Secondly, it is often not necessary to precisely specify the semantics of a service behaviour. For instance, it is often (though not always) sufficient to view a behaviour such as "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 a level of abstraction that is too low. Thirdly, some amount of ambiguity within the confines of reasonable interpretation is often even *desirable* from the point of view of businesses (and in particular their legal departments). The precise behaviour semantics of these languages leaves no wiggle room and may thus actually be a deterrent to adoption.

This paper presents a high-level, generic privacy policy language, S4P, that keeps the semantics of behaviours abstract. Its language design was driven by a small set of design goals to address a number of shortcomings of previous languages:

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 PII disclosure, namely user preferences and service policies, and enable satisfaction checking between the two.

3. It should support, and distinguish between, both permissions and promises over service behaviours, in both user preferences and service policies.

4. As usability, and readability in particular [24], 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, supporting parameterized behaviours and data types, transitive and other recursive conditions and arbitrary constraints.

6. It should support credential-based delegation of authority, which is crucial for modern decentralised and distributed architectures [12].

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. In other words, concrete behaviour ontologies and semantics can be plugged into the language in a modular fashion according to need. Furthermore, 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, or by dynamic monitoring, or static analysis.

At first sight, one might think that, due to the high abstractness of the language, not much insight can be gained with regards to privacy. We show that this is not the case. In particular, our contributions, apart from language design, include the following.

- 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 4). 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?"

- An operational semantics that formalizes three protocols: one for disclosing personal data, one that additionally allows policy evolution, and one that allows services to forward user data to third parties (Section 5). This answers the question: "how can S4P be deployed in a network of collaborating agents to ensure safe communication?" The formalization enables us to state and prove useful safety properties of the protocols, despite the language's abstractness.

Related work is discussed in Section 2. Section 6 concludes the paper with a discussion on implementation, performance and expressiveness issues. Appendix A contains an extended example of a S4P preference and policy.

2 Related work

P3P [14] 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 [15]) for managing preferences and checking them against policies are required. The downside of this approach is that the semantic correspondence between preferences and P3P policies is unclear. 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 (*cf.* [20]). P3P does not satisfy any of the six design goals in Section 1.

DAMP [6] 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 (not discussed in this paper). 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.* [3] 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 (*cf.* [13]), and a user's release policy (corresponding to "preference" in our terminology) cannot express requirements on the service's privacy promises. Like P3P, the language is tied to a predefined vocabulary, lacks a formal semantics, and does not satisfy any of the design goals from Section 1.

Simple Privacy Language (SIMPL) [27] is a restricted subset of English for specifying both preferences and policies. The language is equipped with a small set of specific constructs such as constraining the "verification level" of services or deleting data within some time. Its expressiveness is thus rather limited, but it does satisfy design goals 2, 3 and 4 from Section 1. Only a small part of SIMPL's model-theoretic semantics is currently formally defined.

Barth *et al.* [5] use linear temporal logic to specify positive and negative temporal constraints on the global trace of a network of principals sending user data between each other. 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 (in particularly not non-monotonic actions such as deletion), and extensions would be non-trivial as the effects of behaviours on the state are modelled explicitly. Their formalism could be seen as an example of how temporal logic could be employed to elegantly model our abstract behaviours on a more detailed level, if required.

EPAL [4] is a language for specifying and enforcing organizations' internal rules for accessing user data; essentially, it is an access control language (comparable to XACML [29]) with a privacy-centric vocabulary. It does not deal with specifying user preferences and matching them against policies.

3 S4P Syntax and Query Evaluation

This section defines the syntax and query evaluation semantics of S4P.

Informal overview. In an *encounter* between a user and a service, the service requests some personally identifiable information (PII[1]) from the user, and the user may agree or disagree to the disclosure. Disclosure depends on the service's properties and its privacy *policy*, a document that details how the service is going to handle users' data. To automate this decision process, the policy is written in S4P, a formal language that machines can interpret. Furthermore, the user also has a document written in S4P, called *preference*, which specifies her requirements on the service's properties and on its policy for this encounter.

We assume that there is a predefined collection of parameterised PII-relevant service *behaviours*, and a corresponding vocabulary for representing these behaviours. This vocabulary generally depends on the application domain and may include phrases such as "use `Email` for `Stats`", "delete `Email` within `13 days`", and "have procedures to `label records`". S4P is not tied to a particular set of behaviours; rather, these can be plugged into the language depending on the application domain.

Both policies and preferences consist of a set of *assertions* and a *query*. Assertions in a preference express what a service *may*, or is permitted to, do with the user's PII. They make use of the special modal verb "may", and are thus also called may-assertions. In other words, they specify an *upper bound* on a service's behaviours with respect to the PII. In the may-assertions below, the

[1]For our purposes, the term PII can actually be rather loosely interpreted as any kind of valuable private data that a user possesses and that may be disclosed to services. By a slight abuse of grammar, we will use the plural "PIIs" to mean "pieces of personally identifiable information".

	User Preference	**Service Policy**
Permissions (upper bounds)	may-assertions User gives permissions	may-query Service asks for permissions
Promises (lower bounds)	will-query User asks for promises	will-assertions Service gives promises

Figure 1: Dualities of may and will assertions and queries.

user Alice permits booking services to use her email address for any purpose apart from marketing and statistics, and allows any service to not retain her email addresses. During an encounter, the *placeholder* ⟨Svc⟩ gets dynamically instantiated with the identity of the service.

(U1) $\texttt{Alice}$ says ⟨Svc⟩ may use $\texttt{Email}$ for *purp* if ⟨Svc⟩ is a $\texttt{BookingSvc}$
where $purp \notin \{\texttt{Marketing}, \texttt{Stats}\}$

(U2) $\texttt{Alice}$ says ⟨Svc⟩ may delete $\texttt{Email}$ within t

The second part of any preference is a will-query, which specifies a *lower bound* on a service's properties and behaviours. In other words, it expresses *obligations*, i.e. the behaviours that a service must exhibit. Alice's will-query (UQ) below requires services to delete her email address within 30 days.

(UQ) $\exists t$ (⟨Svc⟩ says ⟨Svc⟩ will delete $\texttt{Email}$ within t? $\wedge$ $t \leq$ 30 days?)

Assertions in a policy express what a service *will* certainly do, or promises to do, with the user's PII. They make use of the special modal verb "will", and are thus also called will-assertions. In other words, they specify a *lower bound* on a service's behaviours with respect to the PII. In the following will-assertion, eBooking promises to delete email addresses within 7 days.

(S1) eBooking says eBooking will delete Email within 7 days

The second part of any policy is a may-query, which specifies an *upper bound* on a service's behaviours. In other words, it expresses and advertises *all possible* relevant behaviours of the service. In the may-query below, eBooking asks for permission to use the collected email address for sending out newsletters. The *placeholder* ⟨Usr⟩ gets instantiated with the user's identity during an encounter.

(SQ) ⟨Usr⟩ says eBooking may use Email for News? $\wedge$
⟨Usr⟩ says eBooking may delete Email within 7 days?

With this may-query, the service declares that it will not exhibit any other relevant behaviours than the specified ones. Fig. 1 summarises the dualities between preference and policy, permissions and promises, assertions and queries, and the may and will modalities.

Appendix A contains a longer example of a preference and a policy.

Preference/policy syntax. 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, for example, include arithmetic operations and relations, regular expressions and environmental constraints.

An *assertion* α is of the form $\langle E \textsf{ says } f_0 \textsf{ if } f_1, \ldots, f_n \textsf{ where } c\rangle$, where E is a constant[2] from **Const**, the f_i are *facts* (defined below), and c is a *constraint* on variables occurring in the assertion. In an assertion $\alpha = \langle e \textsf{ says } f \textsf{ if } f_1, \ldots, f_n \textsf{ where } c\rangle$, the keyword "if" is omitted when $n = 0$; likewise, "where c" is omitted when $c = \textsf{true}$.

Henceforth, we keep to the following conventions for metavariables: 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).

We can now define the syntax of *facts* f:

$$
\begin{array}{llll}
\text{Fact} & f & ::= & a \\
 & & | & e \textsf{ can say } f \\
 & & | & e \textsf{ may } b \\
 & & | & e \textsf{ will } b
\end{array}
$$

The syntax of *queries* q is defined as follows.

$$
\begin{array}{llll}
\text{Query} & q & ::= & e \textsf{ says } f? \\
 & & | & c? \\
 & & | & \neg q \\
 & & | & q_1 \wedge q_2 \\
 & & | & q_1 \vee q_2 \\
 & & | & \exists x(q)
\end{array}
$$

[2]Intuitively, E is of type user or service, but it is not necessary to formally distinguish constant types (such as user, service, PII category) until Section 5, even though we do use them informally.

Facts with $\textsf{can say}$ are used to express *delegation of authority* and have a special query evaluation semantics, as defined in the proof system below. Facts involving $\textsf{may}$ and $\textsf{will}$ are not treated special by query evaluation, but are essential for the privacy-related model semantics in Section 4.

Since many of our further definitions are parameterized by the encounter, we introduce the following definition.

Definition 3.1. A user-service pair is a pair of a user name and a service name during an encounter. Formally: a *user-service* pair $\tau = (U, S)$ is a pair of constants.

Recall that the lower bound on service behaviours specified in users' preferences and the upper bound specified in services' policies are expressed as a $\textsf{will}$-query and a $\textsf{may}$-query, respectively, as defined below.

Definition 3.2. Let $\tau = (U, S)$ be a user-service pair.

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

The definition above syntactically restricts the query occurring in a policy or a preference to those that can be given an intuitive meaning in terms of an upper or a lower bound on behaviours, such that the formal query evaluation semantics described later in this section matches this meaning. Disjunction and, similarly, existential quantification are allowed and have an obvious intuitive meaning within a $\textsf{will}$-query, e.g.

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

A $\textsf{may}$-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:

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

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

We can now define the syntax of preferences and policies.

Definition 3.3. 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-$\textsf{will}$-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-$\textsf{may}$-query.

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 slightly simplified variant of the one from SecPAL [8]. We first define a two-rule proof system that generates ground judgements of the form $\mathcal{A} \vdash E\ \mathsf{says}\ F$:

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

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

Example. Continuing the example from above, suppose `Alice` has a third preference assertion

(U3) `Alice` says `CA` can say x is a `BookingSvc`,

and `eBooking` has a credential (i.e., a digitally signed assertion) issued by `CA`:

(S2) `CA` says `eBooking` is a `BookingSvc`.

Then if $\mathcal{A}$ is the set of assertions (U1, 2, 3) and (S1, 2) (with ⟨Usr⟩ instantiated to `Alice` and ⟨Svc⟩ to `eBooking`), we can prove $\mathcal{A} \vdash$ `Alice` says `eBooking` may use `Email` for `News`, since (U3) and (S2) together imply `Alice` says `eBooking` is a `BookingSvc`.

In the example, `Alice` uses (U3) to *delegate authority* over `BookingSvc` role membership to `CA`. Authority delegation has long been recognized as an essential feature in decentralized authorization and access control. Although delegation is just as important in the privacy context, it has not been supported in any previous privacy language.

Compound queries. The relation $\vdash$ so far only deals with the case where the query is of the basic form $\langle e\ \mathsf{says}\ f? \rangle$. We extend it to all closed queries in a straightforward way, 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.

Example. Simplifying (we will speak about placeholders later), the query (SQ) is evaluated by separately evaluating all conjuncts and checking that each of them is true.

Satisfaction. Should a user agree to the disclosure of her PII? This depends on whether the service's policy *satisfies* her preference. Checking that a policy satisfies a preference consists of two steps. Firstly, every behaviour declared as *possible* in the policy must be *permitted* by the preference. Therefore, 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. Secondly, every behaviour declared as *obligatory* in the preference must be *promised* by the policy. Therefore, 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 (*cf.* Fig. 1), checking if a service policy satisfies a user preference is straightforward. 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 have to be evaluated against the union of the assertions in the policy *and* the preference.

Definition 3.4. A τ-policy $\Pi_{\mathrm{pl}} = (\mathcal{A}_{pl}, q_m)$ *satisfies* a τ-preference $\Pi_{\mathrm{pr}} = (\mathcal{A}_{pr}, q_w)$ iff $\mathcal{A}_{pl} \cup \mathcal{A}_{pr} \vdash q_m \wedge q_w$.

Example. The (`Alice`,`eBooking`)-policy consisting of assertions $(\mathsf{S1}, 2)$ and query (SQ) satisfies the preference consisting of assertions $(\mathsf{U1}, 2, 3)$ and query (UQ), because both queries are derivable from the union of the assertions. □

We assume that on an encounter between U and S, user U provides a (U, S)-preference and service S provides a (U, S)-policy. In practice, preferences and policies are written with *placeholders* that get instantiated when the encounter is initiated, with values that are specific to the encounter. In particular, the concrete syntax may include $\langle \mathsf{Usr} \rangle$ and $\langle \mathsf{Svc} \rangle$ that get instantiated with U and S, respectively. The formal sections in this paper assume that such placeholders in preferences and policies have all been instantiated.

Def. 3.4 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*. Indeed, there is as yet no definition of what we mean by "correct". The following section formalizes a notion of correctness and proves correctness of the satisfaction checking procedure.

4 Trace Semantics

Policies and preferences specify upper and lower bounds on services' behaviours. So what we are interested in is whether a particular run, or *trace*, of a service *complies* with a policy or a preference. Since we are only interested in which PII-relevant behaviours a trace exhibits, 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 $\mathbf{Beh}(t)$. (And conversely, every ground behaviour atom can be seen as a trace property.)

Definition 4.1. 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 $T_1 \subseteq T_2$.

In the following, we define a model-theoretic semantics for S4P policies and preferences by mapping them to *sets* of traces. Furthermore, we will show that

$\mathcal{B} \models^{\text{wa}}_{\tau,\mathcal{A}} \mathcal{A}_{pl}$	Behaviours $\mathcal{B}$ include all the promises made by the will-assertions in $\mathcal{A} \cup \mathcal{A}_{pl}$.
$\mathcal{B} \models^{\text{mq}}_{\tau,\mathcal{A}} q_m$	Behaviours $\mathcal{B}$ are contained in the behaviours for which permission is asked for in the τ-may-query q_m, in the context of $\mathcal{A}$.
$\mathcal{B} \models^{\text{ma}}_{\tau,\mathcal{A}} \mathcal{A}_{pr}$	Behaviours $\mathcal{B}$ are contained in the behaviours permitted by the may-assertions in $\mathcal{A} \cup \mathcal{A}_{pr}$.
$\mathcal{B} \models^{\text{wq}}_{\tau,\mathcal{A}} q_w$	Behaviours $\mathcal{B}$ include all the obligations required by the τ-will-query q_w, in the context of $\mathcal{A}$.
$[\![\Pi_{\text{pl}}]\!]^{\text{pl}}_{\tau,\mathcal{A}}$	Set of all traces that comply with policy Π_{pl}, in the context of $\mathcal{A}$.
$[\![\Pi_{\text{pr}}]\!]^{\text{pr}}_{\tau,\mathcal{A}}$	Set of all traces that comply with preference Π_{pr}, in the context of $\mathcal{A}$.

Figure 2: Overview of the formal notation in Section 4

if a policy satisfies a preference, then the set of traces induced by the policy is at least as strict as the set induced by the preference. Consequently, checking that a policy satisfies a preference is sufficient for proving that the trace exhibited by the service complies with the user preference, assuming that the trace also complies with the service's own policy. This follows from the simple lemma below:

Lemma 4.2. Let t be a trace and T_1, T_2 be sets of traces. If t complies with T_1 and T_1 is at least as strict as T_2 then t also complies with T_2.

Fig. 2 provides an overview of the formal notation introduced in this section.

4.1 Trace Semantics of Policies

We first define two auxiliary relations that are used for specifying the trace semantics of a policy.

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 $\mathcal{B} \models^{\text{wa}}_{\tau,\mathcal{A}} \mathcal{A}_{pl}$ holds if the behaviours in $\mathcal{B}$ include all behaviours promised by will-assertions in $\mathcal{A}_{pl}$, together with the foreign assertions $\mathcal{A}$ in the context (later, $\mathcal{A}$ will be instantiated to the assertions from the user preference):

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

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 $\mathcal{B} \models^{\text{mq}}_{\tau,\mathcal{A}} q_m$ 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 be instantiated to the assertions from both the service policy and the user preference). The relation is defined as the smallest relation satisfying:

$$\mathcal{B} \models^{\text{mq}}_{\tau,\mathcal{A}} U \mathsf{\ says\ } S \mathsf{\ may\ } B? \quad \text{if } \mathcal{B} \subseteq \{B\}$$

$$\mathcal{B} \models^{\text{mq}}_{\tau,\mathcal{A}} q_1 \wedge q_2 \quad \text{if there exist } \mathcal{B}_1, \mathcal{B}_2 \text{ such that } \mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2 \text{ and } \mathcal{B}_1 \models^{\text{mq}}_{\tau,\mathcal{A}} q_1 \text{ and } \mathcal{B}_2 \models^{\text{mq}}_{\tau,\mathcal{A}} q_2$$

$$\emptyset \models^{\text{mq}}_{\tau,\mathcal{A}} q \quad \text{if } \mathcal{A} \vdash q \text{ and no subquery of the form } \langle U \mathsf{\ says\ } S \mathsf{\ may\ } B? \rangle \text{ occurs in } q$$

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

Definition 4.3. Let $\tau = (U, S)$, $\Pi_{\text{pl}} = (\mathcal{A}_{pl}, q_m)$ be a τ-policy, and $\mathcal{A}$ a set of assertions. Then $[\![\Pi_{\text{pl}}]\!]^{\text{pl}}_{\tau,\mathcal{A}}$ denotes the set of all traces t such that

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

Example. If $\tau = (\texttt{Alice}, \texttt{eBooking})$, Π_{pl} consists of assertions (S1, 2) and query (SQ), and $\mathcal{A}$ consists of assertions (U1, 2, 3) (with placeholders instantiated by τ), then $[\![\Pi_{\text{pl}}]\!]^{\text{pl}}_{\tau,\mathcal{A}}$ denotes the set of all traces t such that

$$\{\mathsf{delete}\ \texttt{Email}\ \mathsf{within}\ \texttt{7 days}\} \subseteq \mathbf{Beh}(t) \subseteq \{\mathsf{delete}\ \texttt{Email}\ \mathsf{within}\ \texttt{7 days}, \mathsf{use}\ \texttt{Email}\ \mathsf{for}\ \texttt{News}\}.$$

4.2 Trace Semantics of Preferences

Again, we first define two 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 $\mathcal{B} \models^{\text{ma}}_{\tau,\mathcal{A}} \mathcal{A}_{pr}$ holds if all behaviours in $\mathcal{B}$ are contained in the set of behaviours permitted by the may-assertions in $\mathcal{A}_{pr}$, together with the foreign assertions $\mathcal{A}$ in the context (later, $\mathcal{A}$ will be instantiated to the assertions from the service policy):

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

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 $\mathcal{B} \models^{\text{wq}}_{\tau,\mathcal{A}} q_w$ 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 be instantiated to the assertions from both the service policy and the user preference). The relation is defined as the smallest relation satisfying the following:

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

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

Definition 4.4. Let $\tau = (U, S)$ be a user-service pair, $\Pi_{\text{pr}} = (\mathcal{A}_{pr}, q_w)$ a preference, and $\mathcal{A}$ a set of assertions. Then $[\![\Pi_{\text{pr}}]\!]^{\text{pr}}_{\tau,\mathcal{A}}$ denotes the set of all traces t such that

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

Example. If $\tau = (\texttt{Alice}, \texttt{eBooking})$, Π_{pr} consists of assertions $(\textsf{U1}, 2, 3)$ and query $(\textsf{UQ})$ and $\mathcal{A}$ consists of assertions $(\textsf{S1}, 2)$ (with placeholders instantiated by τ), then $[\![\Pi_{\text{pr}}]\!]^{\text{pr}}_{\tau,\mathcal{A}}$ denotes the set of all traces t such that $\langle \textsf{delete}\ \texttt{Email}\ \textsf{within}\ T\rangle \in \textbf{Beh}(t)$ for some constant $T \leq \textsf{30 days}$ and $\textbf{Beh}(t) \subseteq \{\textsf{delete}\ \texttt{Email}\ \textsf{within}\ T' \mid T' \in \textbf{Const}\} \cup \{\textsf{use}\ \texttt{Email}\ \textsf{for}\ P \mid P \neq \texttt{Marketing} \wedge P \neq \texttt{Stats}\}$.

4.3 Satisfaction and Compliance

Our main correctness theorem, Theorem 4.8, is based on two lemmas. Lemma 4.5 shows that checking a policy's may-query against a set of assertions is sufficient for guaranteeing that a set of behaviours respects the upper bound specified by the assertions, given that the set of behaviours also respects the upper bound specified by the may-query. Similarly, Lemma 4.6 shows that checking a preference's will-query against a set of assertions guarantees that a set of behaviours respects the lower bound specified by the will-query, given that the set of behaviours also respects the lower bound specified by the assertions.

Lemma 4.5. Let $\mathcal{A}$ be a set of assertions, q_m a closed τ-may-query, and $\mathcal{B}$ a set of ground behaviour atoms. If $\mathcal{A} \vdash q_m$ and $\mathcal{B} \models^{\text{mq}}_{\tau,\mathcal{A}} q_m$ then $\mathcal{B} \models^{\text{ma}}_{\tau,\mathcal{A}} \mathcal{A}$.

Lemma 4.6. Let $\mathcal{A}$ be a set of assertions, q_w a closed τ-will-query, and $\mathcal{B}$ a set of ground behaviour atoms. If $\mathcal{A} \vdash q_w$ and $\mathcal{B} \models^{\text{wa}}_{\tau,\mathcal{A}} \mathcal{A}$ then $\mathcal{B} \models^{\text{wq}}_{\tau,\mathcal{A}} q_w$.

Based on these two lemmas, Lemma 4.7 states that checking that the policy satisfies the preference (by checking that all queries are successfully evaluated) is sufficient for guaranteeing that the set of traces represented by the policy is at least as strict as the set of traces represented by the preference.

Lemma 4.7. Let $\Pi_{\text{pl}} = (\mathcal{A}_{pl}, q_m)$ be a τ-policy and $\Pi_{\text{pr}} = (\mathcal{A}_{pr}, q_w)$ a τ-preference. If Π_{pl} satisfies Π_{pr} then $[\![\Pi_{\text{pl}}]\!]^{\text{pl}}_{\tau,\mathcal{A}_{pr}}$ is at least as strict as $[\![\Pi_{\text{pr}}]\!]^{\text{pr}}_{\tau,\mathcal{A}_{pl}}$.

Theorem 4.8 is a corollary of Lemma 4.7 and links up the proof-theoretic notion of satisfaction with the model-theoretic notion of compliance: assuming that a service trace complies (*cf.* Def. 4.1) with the service's own policy, successfully evaluating all queries is sufficient for guaranteeing that the trace also complies with the preference.

Theorem 4.8. Let $\Pi_{\text{pl}} = (\mathcal{A}_{pl}, q_m)$ be a τ-policy and $\Pi_{\text{pr}} = (\mathcal{A}_{pr}, q_w)$ a τ-preference. If a trace t complies with $[\![\Pi_{\text{pl}}]\!]^{\text{pl}}_{\tau,\mathcal{A}_{pr}}$ and Π_{pl} satisfies Π_{pr}, then t complies with $[\![\Pi_{\text{pr}}]\!]^{\text{pr}}_{\tau,\mathcal{A}_{pl}}$.

Note that this theorem is completely independent of any concrete instantiation of the behaviours, and of the **Beh** mapping in particular. We are thus able to prove the essential correctness property for S4P despite its abstractness. Of course, if behaviour-specific properties are to be proved, then **Beh** needs to be filled with some structure. We show an example of such a partial instantiation of **Beh** in Section 5.1.2.

5 Protocol for 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.

We describe the protocol on a high level, abstracting away from the internal details of the services and of the communication. However, our description presents an example of how the a notion of a trace from S4P can be instantiated.

Informally, we fix an arbitrary run of a set of services and users and show the conditions that the protocol imposes on the services in the run. The protocol allows the services to operate with PIIs only in the following ways:

- send PIIs to services;
- receive PIIs from services and users;
- change policies on-the-fly.

(Incorporating additional actions, like forgetting PIIs, into the protocol leads to similar safeness results; for simplicity we are keeping the protocol small in the current presentation.)

5.1 Informal protocol description

Now we informally describe what actions a service may execute and how.

A service may involve in an encounter with a user, it may involve in an encounter with another services, it may alter its own policy or execute an internal action.

We assume that a service attaches a distinct preference and a distinct policy to each received PII of each user. On internal events, the service is required to keep preferences, policies and stored PIIs unchanged. Now we describe the communication events and policy evolutions.

5.1.1 User-service encounter

Now we show how a user discloses a PII to a service.

If a service S wishes to collect a PII from a user U, then the following steps should happen (here, $\tau = (U, S)$):

- U and S decide on a τ-preference Π_{pr} and a τ-policy Π_{pl}, respectively, to be used for this encounter. The protocol allows any way of obtaining the policy and the preference: these may be fixed or result from some automated or manual negotiation, e.g. U may use checkboxes in a graphical user interface to customize a protocol template provided by S.
- If Π_{pl} satisfies Π_{pr}, then U sends PII to S, otherwise the protocol is aborted. The protocol allows any trusted way of checking satisfaction. The trust relations should dictate who checks satisfaction:
  - U, as the main stakeholder, wishing to keep even the preference secret to a widest possible extent;
  - or S, wishing to keep those parts of its policy secret which are irrelevant for the current encounter;
  - or a trusted third party.

  From the viewpoint of maintaining privacy U is the safest choice.
- S keeps a copy of Π_{pl} and Π_{pr} together with the PII.

Real-world devices have limited resources. For example, mobile devices are restricted in the size of the fetched policies, in the ability to customize them and in the computation time, which might require S or a third party to provide a fixed policy and check satisfaction. Congested services may not want to compute satisfaction checks, shifting this task to U or to a third party. Communication with distant third parties incurs additional delays, leaving the task to U or S (or a third party on the communication channel between U and S). Relying on trust relations gives us freedom to choose the most adequate way of determining the preference and the policy and the cheapest way of checking satisfaction.

The scheme where the only actions are internal ones and user-service disclosures guarantees that at any point of time, if a service possesses a PII of the user, then the user must have disclosed it to the service in the past, and the service associated the PII with the policy and the preference during the encounter. Furthermore, assuming that the service complies with its policy, it also complies with the preference.

5.1.2 Transitive service-service encounter

Now we show how a service discloses a PII to another service.

Once a PII has been collected by a service, it may be sent on to third-party services, which may in turn disclose the PII further. In most scenarios, disclosing a PII P to a third party S' represents a privacy-relevant behaviour, which should be controlled by preferences and policies. If third-party disclosures should be controlled, atoms of the form $\langle \texttt{send } P \texttt{ to } S' \rangle$ (like $\langle \texttt{send Email to eMarketing} \rangle$)

should be among the behavioural atoms and **Beh** should keep track of those atoms.

Many privacy languages like P3P/APPEL allow preferences to check if a service $S1$ who initially received a PII wants to share it with third parties like $S2$. But those languages don't control the disclosures made by $S2$, which is in most scenarios privacy-relevant. Our protocol controls disclosures along an arbitrarily long chain of services.

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 the preference of U. Again, the protocol allows any trusted way of checking satisfaction. The trust relations should dictate who check satisfaction:
    - U, who is the main stakeholder;
    - or S, who may not trust S' on checking;
    - or S', who may not trust S on handling the policy of S'
    - or a trusted third party.

   Although from the viewpoint of maintaining privacy U is the safest choice, U can't be expected to be always physically available. Thus S is the best choice from the viewpoints of privacy and responsiveness.

As before, limited computational resources influence the choice of place to check satisfaction. The network architects are free to choose any trusted place, fully controlling the trade-off between privacy and available resources like time, space and bandwidth.

Disclosing a PII without knowing the future chain of recipients is perfectly alright in theory: the informal "the service should revoke the user's cookies within 2 years" is expressible as a set of queries "U says S will `revoke cookies within 2 yr`?" for all user identifiers U and all service identifiers S. However, if the set of principals is finite but large or infinite (to model new principals appearing in the course of the time), naively maintaining such a set of queries (i.e. storing a plain text query for each U and S) is infeasible in the finite case and impossible in the infinite case.

The aforementioned *placeholders* $\langle\mathsf{Usr}\rangle$ and $\langle\mathsf{Svc}\rangle$ resolve the issue: the above set of queries is stored and transmitted as a single sentence "$\langle\mathsf{Usr}\rangle$ says $\langle\mathsf{Svc}\rangle$ will `revoke cookies within 2 yr`?". The placeholders are instantiated during encounters by the current user-service pair just before checking satisfaction. Besides keeping the instantiated policy and preference attached to the stored PII of the user, the recipient S must retain the original, uninstantiated "sticky" preference template along with the PII, so that it can later be instantiated using $\tau' = (U, S')$ when S prepares to disclose the PII to S'. Simplifying, we ignore placeholders in the formal model.

The scheme where the only actions are the internal ones and the user-service and service-service disclosures guarantees that at any point of time, if a service S possesses a user's PII, then S either got it from the user directly (as before), or S obtained it in the past via a third-party exchange from some service *Sndr* which possessed the PII at that time. In the latter case, assuming that *Sndr* complies with its own policy, the may-assertions in the user's preference permit the third-party disclosure by *Sndr* represented by the $\langle\mathsf{send}\ P\ \mathsf{to}\ S\rangle$ behaviour atom. Furthermore, assuming that S complies with its own policy, it will also comply with the user's preference.

5.1.3 Policy evolution

Now we show how a service can change its policy.

A service may wish to alter its policy even after having collected a PII. For example, a service S may want to disclose the PII to a previously unknown third party after the original encounter, even though the behaviour corresponding to the disclosure to that third party was not declared in the may-assertions in the policy of S. Or it may wish *not* to exhibit a behaviour it had previously promised in the will-query of the policy, e.g. not to notify the user despite having promised to notify her.

Strictly speaking, both cases represent compliance violations of the service's own original policy.

However, if the new behaviours still comply with the user's original preferences, architects of a particular network may wish to permit such violations (one can argue, for instance, about amending typos in the policy part which is irrelevant for the user). 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.

Assuming that the only actions are the internal ones, user-service disclosure and policy amendment, this scheme guarantees that at any point of time, if a service possesses a PII of a user, then the user must have disclosed it to the service in the past, and the service associated the PII with some (possibly different from the current) policy and the current preference during the encounter. Furthermore, if the service complies with the current policy, it also complies with the user's preference.

5.1.4 Privacy guarantee in general

Now we show the most general privacy guarantee assuming that all actions are allowed: internal, policy amendment, user-service disclosure and service-service disclosure.

The protocol guarantees that if all services comply with their own policies, then at any point of time, if a service possesses a PII of a user, then

- either the user 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, S complies with the user's preference.

5.2 Formal protocol description

Now we formalize the protocol and prove privacy guarantees.

5.2.1 Preliminaries

The protocol is formalized using the following definitions.

Let **Usr** and **Svc** be mutually disjoint subsets of **Const** and $\mathbf{PII} \subseteq \mathbf{Const}$. A *user identifier* is an element of **Usr**, a *service identifier* is an element of **Svc**, a *principal identifier* is an element of $\mathbf{Usr} \dot{\cup} \mathbf{Svc}$, a *PII category* is an element of **PII**. We redefine a *user-service pair* to be an element of $\mathbf{Usr} \times \mathbf{Svc}$ (all previous claims were of the form "for all $(U, S) \in \mathbf{Const}^2 \ldots$" and thus remain valid).

An *S-state label* for $S \in \mathbf{Svc}$ is a partial function that takes $(U, P) \in \mathbf{Usr} \times \mathbf{PII}$ and returns $(\Pi_{\text{pl}}, \Pi_{\text{pr}})$ where Π_{pl} is a (U, S)-policy and Π_{pr} is a (U, S)-preference.

A *T-transition label* for $T \in \mathbf{Svc} \cup \mathbf{Usr}$ is one of the following pairwise different terms:

- *receive event* $\mathsf{rcv}\langle \mathit{Sndr}, U, P, \Pi_{\text{pl}}, \Pi_{\text{pr}}\rangle$ whenever $T \in \mathbf{Svc}$ where $\mathit{Sndr} \in \mathbf{Usr} \cup \mathbf{Svc}$, $U \in \mathbf{Usr}$, $P \in \mathbf{PII}$, Π_{pl} is a (U, T)-policy, Π_{pr} is an (U, T)-preference;
- *sending event* $\mathsf{snd}\langle \mathit{Rcvr}, U, P, \Pi_{\text{pl}}, \Pi_{\text{pr}}\rangle$ where $\mathit{Rcvr} \in \mathbf{Svc}$, $U \in \mathbf{Usr}$, $P \in \mathbf{PII}$, Π_{pl} is a (U, Rcvr)-policy, Π_{pr} is a (U, Rcvr)-preference;
- *policy changing event* $\text{chg}\langle U, P, \Pi_{\text{pl}}\rangle$ whenever $T \in \mathbf{Svc}$ where $U \in \mathbf{Usr}$, $P \in \mathbf{PII}$, Π_{pl} is a (U, T)-policy;
- *internal event* ϵ.

A *trace* is an infinite sequence of T-transition labels for some fixed principal identifier T. Within the paper, all indices are from the set of non-negative integers $\mathbb{N}_0$. Unless otherwise stated, indexing starts from zero.

This definition exemplary instantiates the previous abstract notion of a trace.

A *single service run* for $S \in \mathbf{Svc}$ is a sequence $\langle \sigma_i, \lambda_i\rangle_{i \geq 0}$ where σ_i is an S-state label and λ_i is an S-transition label ($i \geq 0$). The *k^{th} trace of a single service run R* for $k \in \mathbb{N}_0$ is $\text{trace}_k(\langle \sigma_i, \lambda_i\rangle_{i \geq 0}) = \langle \lambda_i\rangle_{i \geq k}$.

A *single user run* for $U \in \mathbf{Usr}$ is an infinite sequence of U-transition labels.

A *synchronized run* is a function $\tilde{R}$ mapping service identifiers S to single service runs for S and user identifiers U to single user runs for U such that the following synchronization condition holds:
assuming that for each $S \in \mathbf{Svc}$ we have $\text{trace}_0(\tilde{R}(S)) = \langle \lambda_i^S\rangle_{i \geq 0}$ and that for each $U \in \mathbf{Usr}$ we have $\tilde{R}(U) = \langle \lambda_i^U\rangle_{i \geq 0}$, we have for all $k \in \mathbb{N}_0$, Rcvr, Sndr, U, P, Π_{pl} and Π_{pr} that

$$\lambda_k^{\mathit{Rcvr}} = \mathsf{rcv}\langle \mathit{Sndr}, U, P, \Pi_{\text{pl}}, \Pi_{\text{pr}}\rangle \quad \text{iff} \quad \lambda_k^{\mathit{Sndr}} = \mathsf{snd}\langle \mathit{Rcvr}, U, P, \Pi_{\text{pl}}, \Pi_{\text{pr}}\rangle.$$

For the purpose of transitive third-party disclosure (and only for this purpose) we assume that behavioural atoms include $\langle \mathsf{send}\ P\ \mathsf{to}\ S\rangle$ for all $P \in \mathbf{PII}$, $S \in \mathbf{Svc}$.

An *evolving run with transitive disclosures* is a synchronized run $\tilde{R}$ such that for all $S \in \mathbf{Svc}$ with single service runs $(\sigma_i, \lambda_i)_{i \geq 0} = \tilde{R}(S)$ and $k \in \mathbb{N}_0$, either

- λ_k is of the form $\mathsf{rcv}\langle \mathit{Sndr}, U, P, \Pi_{\text{pl}}, \Pi_{\text{pr}}\rangle$ for some Sndr, U, P, Π_{pl}, Π_{pr} such that Π_{pl} satisfies Π_{pr}, and $\sigma_{k+1} = \sigma_k[(U, P) \mapsto (\Pi_{\text{pl}}, \Pi_{\text{pr}})]$, and, if $\mathit{Sndr} \in \mathbf{Usr}$, then $\mathit{Sndr} = U$;
- or λ_k is of the form $\mathsf{snd}\langle \mathit{Rcvr}, U, P, \Pi_{\text{pl}}, \Pi_{\text{pr}}\rangle$ for some Rcvr, U, P, Π_{pl}, Π_{pr} such that there is a (U, S)-policy Π'_{pl} and (U, S)-preference Π'_{pr} such

that $\sigma_k(U,P) = (\Pi'_{\text{pl}}, \Pi'_{\text{pr}})$ and $\langle \mathsf{send}\ P\ \mathsf{to}\ \mathit{Rcvr}\rangle \in \mathbf{Beh}(\text{trace}_k(\tilde{R}(S)))$ and $\sigma_k = \sigma_{k+1}$;

- or λ_k is of the form $\text{chg}\langle U, P, \Pi^{\text{new}}_{\text{pl}}\rangle$ for some U, P, $\Pi^{\text{new}}_{\text{pl}}$ such that there are $\Pi^{\text{old}}_{\text{pl}}$ and Π_{pr} such that $\sigma_k(U,P) = (\Pi^{\text{old}}_{\text{pl}}, \Pi_{\text{pr}})$ and $\Pi^{\text{new}}_{\text{pl}}$ satisfies Π_{pr} and $\sigma_{k+1} = \sigma_k[(U,P) \mapsto (\Pi^{\text{new}}_{\text{pl}}, \Pi_{\text{pr}})]$;
- or $\lambda_k = \epsilon$ and $\sigma_k = \sigma_{k+1}$.

A *basic run* is an evolving run with transitive disclosures in which transition labels of all services are only internal and receiving events.

An *evolving run* is an evolving run with transitive disclosures in which transition labels of all services are only internal, receiving and policy changing events.

A *run with transitive disclosures* is an evolving run with transitive disclosures in which transition labels of all services are only internal, receiving and sending events.

For a single service run $\langle \sigma_i, \lambda_i\rangle_{i\geq 0}$ for $S \in \mathbf{Svc}$, $k \geq 0$, $(U,P) \in \text{dom}(\sigma_k)$, the *receiving position of P for U before k* is

$$\max\{i < k \mid \lambda_i \text{ is of the form } \mathsf{rcv}\langle \ldots, U, P, \ldots, \ldots\rangle\}$$

whenever the maximum is defined; the receiving position is undefined otherwise.

5.2.2 Privacy guarantee

Now we are going to prove preservation of privacy for all four types of runs.

All the theorems follow the same scheme and have similar proofs. We start with the simplest one for basic runs and finish with the most general one for evolving runs with transitive disclosures.

For that, we fix an evolving run with transitive disclosures $\tilde{R}$ such that for all $S \in \mathbf{Svc}$ we have $\tilde{R}(S) = \langle \sigma^S_i, \lambda^S_i\rangle_{i\geq 0}$ and for each $U \in \mathbf{Usr}$ we have $\tilde{R}(U) = \langle \lambda^U_i\rangle_{i\geq 0}$. Let $\tau = (U,S) \in \mathbf{Usr} \times \mathbf{Svc}$, $P \in \mathbf{PII}$, $\Pi_{\text{pl}} = (\mathcal{A}_{\text{pl}}, q_{\text{m}})$ a τ-policy, $\Pi_{\text{pr}} = (\mathcal{A}_{\text{pr}}, q_{\text{w}})$ a τ-preference, $k \geq 0$, $(U,P) \notin \text{dom}(\sigma^S_0)$ and $(\Pi_{\text{pl}}, \Pi_{\text{pr}}) = \sigma^S_k(U,P)$.

Theorem 5.1 (Privacy for basic runs)**.** If $\tilde{R}$ is a basic run, then the receiving position r of P for U before k in $\tilde{R}(S)$ is defined and $\lambda^U_r = \mathsf{snd}\langle S, U, P, \Pi_{\text{pl}}, \Pi_{\text{pr}}\rangle$. Furthermore, if $\text{trace}_r(\tilde{R}(S))$ complies with $[\![\Pi_{\text{pl}}]\!]^{\text{pl}}_{\tau,\mathcal{A}_{\text{pr}}}$, so it does with $[\![\Pi_{\text{pr}}]\!]^{\text{pr}}_{\tau,\mathcal{A}_{\text{pl}}}$.

Proof. Induction on k.

- $k = 0$. We can't simultaneously have $(U,P) \notin \text{dom}(\sigma^S_0)$ and $(\Pi_{\text{pl}}, \Pi_{\text{pr}}) = \sigma^S_k(U,P)$, a contradiction.
- $k > 0$. The transition label of S at position $k-1$ can be of two forms.
  - λ^S_{k-1} is of the form $\mathsf{rcv}\langle \widehat{\mathit{Sndr}}, \widehat{U}, \widehat{P}, \widehat{\Pi_{\text{pl}}}, \widehat{\Pi_{\text{pr}}}\rangle$ for some principal identifier $\widehat{\mathit{Sndr}}$, user identifier $\widehat{U}$, PII category $\widehat{P}$, $(\widehat{U}, S)$-policy $\widehat{\Pi_{\text{pl}}}$ that satisfies the $(\widehat{U}, S)$-preference $\widehat{\Pi_{\text{pr}}}$ and $\sigma^S_k = \sigma^S_{k-1}[(\widehat{U}, \widehat{P}) \mapsto (\widehat{\Pi_{\text{pl}}}, \widehat{\Pi_{\text{pr}}})]$. We have two cases.
    - $(U,P) \neq (\widehat{U}, \widehat{P})$. Apply the induction hypothesis. Notice that the receiving position of P for U before k is the same as before $k-1$.

  * $(U,P) = (\widehat{U},\widehat{P})$. Then $\widehat{\Pi_{\text{pl}}} = \Pi_{\text{pl}}$ and $\widehat{\Pi_{\text{pr}}} = \Pi_{\text{pr}}$ and the receiving position of P for U before k is $r = k-1$. By definition of a synchronized run, $\lambda_r^{\widehat{Sndr}} = \mathsf{snd}\langle S,\widehat{U},\widehat{P},\widehat{\Pi_{\text{pl}}},\widehat{\Pi_{\text{pr}}}\rangle$. Since the run is basic, $Sndr \notin \mathbf{Svc}$, so $Sndr \in \mathbf{Usr}$. By definition of a synchronized run, $Sndr = \widehat{U}$. Thus $\lambda_r^S = \mathsf{rcv}\langle U,U,P,\Pi_{\text{pl}},\Pi_{\text{pr}}\rangle$ and $\lambda_r^U = \mathsf{snd}\langle S,U,P,\Pi_{\text{pl}},\Pi_{\text{pr}}\rangle$. Since Π_{pl} satisfies Π_{pr}, apply Thm. 4.8 to $\text{trace}_r(\tilde{R}(S))$.
- $\lambda_{k-1}^S = \epsilon$. Then $\sigma_{k-1}^S = \sigma_k^S$, apply the induction hypothesis and notice that the receiving position of P for U before $k-1$ is the same as the receiving position of P for U before k.

□

Theorem 5.2 (Privacy for evolving runs)**.** If $\tilde{R}$ is an evolving run, then the receiving position r of P for U before k in $\tilde{R}(S)$ is defined and $\lambda_r^U = \mathsf{snd}\langle S,U,P,\Pi_{\text{pl}}^{\text{orig}},\Pi_{\text{pr}}\rangle$ for some $\Pi_{\text{pl}}^{\text{orig}}$. Moreover, if $\text{trace}_r(\tilde{R}(S))$ complies with $[\![\Pi_{\text{pl}}]\!]_{\tau,\mathcal{A}_{\text{pr}}}^{\text{pl}}$, so it does with $[\![\Pi_{\text{pr}}]\!]_{\tau,\mathcal{A}_{\text{pl}}}^{\text{pr}}$.

Proof. Induction on k.

- $k = 0$. Then σ_0^S is both defined and undefined at (U,P), a contradiction.
- $k > 0$. There are three cases for the event label.
  - λ_{k-1}^S is of the form $\mathsf{rcv}\langle\widehat{Sndr},\widehat{U},\widehat{P},\widehat{\Pi_{\text{pl}}},\widehat{\Pi_{\text{pr}}}\rangle$ for some principal identifier $\widehat{Sndr}$, user identifier $\widehat{U}$, PII category $\widehat{P}$, $(\widehat{U},S)$-policy $\widehat{\Pi_{\text{pl}}}$ that satisfies the $(\widehat{U},S)$-preference $\widehat{\Pi_{\text{pr}}}$ and $\sigma_k^S = \sigma_{k-1}^S[(\widehat{U},\widehat{P}) \mapsto (\widehat{\Pi_{\text{pl}}},\widehat{\Pi_{\text{pr}}})]$. We have two cases.
    * $(U,P) \neq (\widehat{U},\widehat{P})$. Apply the induction hypothesis. Notice that the receiving position of P for U before k is the same as before $k-1$.
    * $(U,P) = (\widehat{U},\widehat{P})$. Then $\widehat{\Pi_{\text{pl}}} = \Pi_{\text{pl}}$ and $\widehat{\Pi_{\text{pr}}} = \Pi_{\text{pr}}$ and the receiving position of P for U before k is $r = k-1$. By definition of a synchronized run, $\lambda_r^{\widehat{Sndr}} = \mathsf{snd}\langle S,\widehat{U},\widehat{P},\widehat{\Pi_{\text{pl}}},\widehat{\Pi_{\text{pr}}}\rangle$. Since the run is evolving, $Sndr \notin \mathbf{Svc}$, so $Sndr \in \mathbf{Usr}$. Thus $\lambda_r^S = \mathsf{rcv}\langle U,U,P,\Pi_{\text{pl}},\Pi_{\text{pr}}\rangle$ and $\lambda_r^U = \mathsf{snd}\langle S,U,P,\Pi_{\text{pl}},\Pi_{\text{pr}}\rangle$. Since Π_{pl} satisfies Π_{pr}, apply Thm. 4.8 to $\text{trace}_r(\tilde{R}(S))$.
  - λ_{k-1}^S is of the form $\text{chg}\langle\widehat{U},\widehat{P},\widehat{\Pi_{\text{pl}}^{\text{new}}}\rangle$ for some $\widehat{U} \in \mathbf{Usr}$, $\widehat{P} \in \mathbf{PII}$ and a $(\widehat{U},S)$-policy $\widehat{\Pi_{\text{pl}}^{\text{new}}}$. There are two cases.
    * $(U,P) \neq (\widehat{U},\widehat{P})$. Then $\sigma_{k-1}^S(U,P) = (\Pi_{\text{pl}},\Pi_{\text{pr}})$. Apply the induction hypothesis. Notice that the receiving position of P for U before k is the same as before $k-1$.
    * $(U,P) = (\widehat{U},\widehat{P})$. By definition of an evolving run, $\Pi_{\text{pl}} = \widehat{\Pi_{\text{pl}}^{\text{new}}}$ and there is a (U,S)-policy $\widehat{\Pi_{\text{pl}}^{\text{old}}}$ and a (U,S)-preference Π_{pr} such that $\sigma_{k-1}^S(U,P) = (\widehat{\Pi_{\text{pl}}^{\text{old}}},\Pi_{\text{pr}})$ and $\widehat{\Pi_{\text{pl}}^{\text{new}}}$ satisfies Π_{pr}. Then Π_{pl} satisfies Π_{pr}. By Thm. 4.8, if $\text{trace}_r(\tilde{R}(S))$ complies with $[\![\Pi_{\text{pl}}]\!]_{\tau,\mathcal{A}_{\text{pr}}}^{\text{pl}}$, it also complies with $[\![\Pi_{\text{pr}}]\!]_{\tau,\mathcal{A}_{\text{pl}}}^{\text{pr}}$. Apply the induction assumption to $k-1$ and $\sigma_{k-1}^S(U,P) = (\widehat{\Pi_{\text{pl}}^{\text{old}}},\Pi_{\text{pr}})$ to obtain $\Pi_{\text{pl}}^{\text{orig}}$

and the receiving position r of P for U before $k-1$. Notice that r is also the receiving position of P for U before k.

  - $\lambda^S_{k-1} = \epsilon$. Then $\sigma^S_{k-1} = \sigma^S_k$, apply the induction hypothesis and notice that the receiving position of P for U before $k-1$ is the same as the receiving position of P for U before k.

□

Theorem 5.3 (Privacy for runs with transitive disclosures)**.** If $\tilde{R}$ is a run with transitive disclosures, then the receiving position r of P for U before k in $\tilde{R}(S)$ is defined and either

- $\lambda^U_r = \mathsf{snd}\langle S, U, P, \Pi_{\mathrm{pl}}, \Pi_{\mathrm{pr}}\rangle$,
- or there is $\mathit{Sndr} \in \mathbf{Svc}$ such that $\lambda^{\mathit{Sndr}}_r = \mathsf{snd}\langle S, U, P, \Pi_{\mathrm{pl}}, \Pi_{\mathrm{pr}}\rangle$ and $\sigma^{\mathit{Sndr}}_r(U, S)$ is of the form $(\Pi'_{\mathrm{pl}}, \Pi'_{\mathrm{pr}})$ for some $\tau' = (U, \mathit{Sndr})$-policy $\Pi'_{\mathrm{pl}} = (\mathcal{A}'_{\mathrm{pl}}, q'_{\mathrm{m}})$ and τ'-preference $\Pi'_{\mathrm{pr}} = (\mathcal{A}'_{\mathrm{pr}}, q'_{\mathrm{w}})$. In this case, if $\mathrm{trace}_r(\tilde{R}(\mathit{Sndr}))$ complies with $[\![\Pi'_{\mathrm{pl}}]\!]^{\mathrm{pl}}_{\tau',\mathcal{A}'_{\mathrm{pr}}}$, then $\{\mathsf{send}\ P\ \mathsf{to}\ S\} \models^{\mathrm{ma}}_{\tau',\mathcal{A}'_{\mathrm{pl}}} \mathcal{A}'_{\mathrm{pr}}$.

Furthermore, if $\mathrm{trace}_r(\tilde{R}(S))$ complies with $[\![\Pi_{\mathrm{pl}}]\!]^{\mathrm{pl}}_{\tau,\mathcal{A}_{\mathrm{pr}}}$, so it does with $[\![\Pi_{\mathrm{pr}}]\!]^{\mathrm{pr}}_{\tau,\mathcal{A}_{\mathrm{pl}}}$.

Proof. We are going to show a slightly stronger property by induction on k. Namely, in addition to the stated claim, we will show that Π_{pl} satisfies Π_{pr}.

- $k = 0$. Then σ^S_0 is both defined and undefined at (U, P), a contradiction.
- $k > 0$. There are three cases for the event label.
  - λ^S_{k-1} is of the form $\mathsf{snd}\langle \widehat{\mathit{Rcvr}}, \widehat{U}, \widehat{P}, \widehat{\Pi_{\mathrm{pl}}}, \widehat{\Pi_{\mathrm{pr}}}\rangle$ for some $\widehat{\mathit{Rcvr}}$, $\widehat{U}$, $\widehat{P}$, $\widehat{\Pi_{\mathrm{pl}}}$, $\widehat{\Pi_{\mathrm{pr}}}$. Then $\sigma^S_{k-1}(U, P) = (\Pi_{\mathrm{pl}}, \Pi_{\mathrm{pr}})$. Apply the induction hypothesis. Then notice that the receiving position before k is the same as receiving position before $k-1$.
  - λ^S_{k-1} is of the form $\mathsf{rcv}\langle \widehat{\mathit{Sndr}}, \widehat{U}, \widehat{P}, \widehat{\Pi_{\mathrm{pl}}}, \widehat{\Pi_{\mathrm{pr}}}\rangle$ for some principal identifier $\widehat{\mathit{Sndr}}$, user identifier $\widehat{U}$, PII category $\widehat{P}$, $(\widehat{U}, S)$-policy $\widehat{\Pi_{\mathrm{pl}}}$ that satisfies the $(\widehat{U}, S)$-preference $\widehat{\Pi_{\mathrm{pr}}}$ and $\sigma^S_k = \sigma^S_{k-1}[(\widehat{U}, \widehat{P}) \mapsto (\widehat{\Pi_{\mathrm{pl}}}, \widehat{\Pi_{\mathrm{pr}}})]$. We have two cases.
    * $(U, P) \neq (\widehat{U}, \widehat{P})$. Apply the induction hypothesis. Notice that the receiving position of P for U before k is the same as before $k-1$.
    * $(U, P) = (\widehat{U}, \widehat{P})$. Then $\widehat{\Pi_{\mathrm{pl}}} = \Pi_{\mathrm{pl}}$ and $\widehat{\Pi_{\mathrm{pr}}} = \Pi_{\mathrm{pr}}$ and the receiving position of P for U before k is $r = k-1$. In particular, Π_{pl} satisfies Π_{pr}. By Thm. 4.8, if $\mathrm{trace}_r(\tilde{R}(S))$ complies with $[\![\Pi_{\mathrm{pl}}]\!]^{\mathrm{pl}}_{\tau,\mathcal{A}_{\mathrm{pr}}}$, it also complies with $[\![\Pi_{\mathrm{pr}}]\!]^{\mathrm{pr}}_{\tau,\mathcal{A}_{\mathrm{pl}}}$. There are two possibilities for $\widehat{\mathit{Sndr}}$.
      · $\widehat{\mathit{Sndr}} \in \mathbf{Usr}$. By definition of a run with transitive disclosures we have $\widehat{\mathit{Sndr}} = \widehat{U}$, so $\widehat{\mathit{Sndr}} = U$. By definition of a synchronized run, $\lambda^U_{k-1} = \mathsf{snd}\langle S, U, P, \Pi_{\mathrm{pl}}, \Pi_{\mathrm{pr}}\rangle$.
      · $\widehat{\mathit{Sndr}} \in \mathbf{Svc}$. Let $\mathit{Sndr} = \widehat{\mathit{Sndr}}$. By definition of a synchronized run, $\lambda^{\mathit{Sndr}}_{k-1} = \mathsf{snd}\langle S, U, P, \Pi_{\mathrm{pl}}, \Pi_{\mathrm{pr}}\rangle$. Let $\tau' = (U, \mathit{Sndr})$. By definition of a run with transitive disclosures there is a τ'-

policy $\Pi'_{\mathrm{pl}} = (\mathcal{A}'_{\mathrm{pl}}, q'_{\mathrm{m}})$ and a τ'-preference $\Pi'_{\mathrm{pr}} = (\mathcal{A}'_{\mathrm{pr}}, q'_{\mathrm{w}})$ such that $\sigma^{Sndr}_{k-1}(U,P) = (\Pi'_{\mathrm{pl}}, \Pi'_{\mathrm{pr}})$ and $\langle \texttt{send}\ P\ \texttt{to}\ S\rangle \in \mathbf{Beh}(t')$ where $t' = \mathrm{trace}_{k-1}(\tilde{R}(Sndr))$. By induction hypothesis, Π'_{pl} satisfies Π'_{pr}. Assume that t' complies with $[\![\Pi'_{\mathrm{pl}}]\!]^{\mathrm{pl}}_{\tau',\mathcal{A}'_{\mathrm{pr}}}$. By Thm. 4.8, t' complies with $[\![\Pi'_{\mathrm{pr}}]\!]^{\mathrm{pr}}_{\tau',\mathcal{A}'_{\mathrm{pl}}}$. By Def. 4.4, $\mathbf{Beh}(t') \models^{\mathrm{ma}}_{\tau',\mathcal{A}'_{\mathrm{pl}}} \mathcal{A}'_{\mathrm{pr}}$. By definition of permissions allowed by preferences, $\mathbf{Beh}(t') \subseteq \{B \mid \mathcal{A}'_{\mathrm{pr}} \cup \mathcal{A}'_{\mathrm{pl}} \vdash U\ \mathsf{says}\ Sndr\ \mathsf{may}\ B\}$. Thus $\{\langle \texttt{send}\ P\ \texttt{to}\ S\rangle\} \subseteq \{B \mid \mathcal{A}'_{\mathrm{pr}} \cup \mathcal{A}'_{\mathrm{pl}} \vdash U\ \mathsf{says}\ Sndr\ \mathsf{may}\ B\}$. So $\{\langle \texttt{send}\ P\ \texttt{to}\ S\rangle\} \models^{\mathrm{ma}}_{\tau',\mathcal{A}'_{\mathrm{pl}}} \mathcal{A}'_{\mathrm{pr}}$.

– $\lambda^S_{k-1} = \epsilon$. Then $\sigma^S_{k-1} = \sigma^S_k$, apply the induction hypothesis and notice that the receiving position of P for U before $k-1$ is the same as the receiving position of P for U before k.

□

Theorem 5.4 (Privacy for evolving runs with transitive disclosures)**.** If $\tilde{R}$ is an evolving run with transitive disclosures, then the receiving position r of P for U before k in $\tilde{R}(S)$ is defined and either

- $\lambda^U_r = \mathsf{snd}\langle S, U, P, \Pi^{\mathrm{orig}}_{\mathrm{pl}}, \Pi_{\mathrm{pr}}\rangle$ for some $\Pi^{\mathrm{orig}}_{\mathrm{pl}}$,
- or there is $Sndr \in \mathbf{Svc}$ such that $\lambda^{Sndr}_r = \mathsf{snd}\langle S, U, P, \Pi^{\mathrm{orig}}_{\mathrm{pl}}, \Pi_{\mathrm{pr}}\rangle$ and $\sigma^{Sndr}_r(U,S)$ is of the form $(\Pi'_{\mathrm{pl}}, \Pi'_{\mathrm{pr}})$ for some $\tau' = (U, Sndr)$-policy $\Pi'_{\mathrm{pl}} = (\mathcal{A}'_{\mathrm{pl}}, q'_{\mathrm{m}})$ and τ'-preference $\Pi'_{\mathrm{pr}} = (\mathcal{A}'_{\mathrm{pr}}, q'_{\mathrm{w}})$. In this case, if $\mathrm{trace}_r(\tilde{R}(Sndr))$ complies with $[\![\Pi'_{\mathrm{pl}}]\!]^{\mathrm{pl}}_{\tau',\mathcal{A}'_{\mathrm{pr}}}$, then $\{\mathsf{send}\ P\ \mathsf{to}\ S\} \models^{\mathrm{ma}}_{\tau',\mathcal{A}'_{\mathrm{pl}}} \mathcal{A}'_{\mathrm{pr}}$.

Moreover, if $\mathrm{trace}_r(\tilde{R}(S))$ complies with $[\![\Pi_{\mathrm{pl}}]\!]^{\mathrm{pl}}_{\tau,\mathcal{A}_{\mathrm{pr}}}$, so it does with $[\![\Pi_{\mathrm{pr}}]\!]^{\mathrm{pr}}_{\tau,\mathcal{A}_{\mathrm{pl}}}$.

Proof. We are going to show a slightly stronger property by induction on k. Namely, in addition to the stated claim, we will show that Π_{pl} satisfies Π_{pr}.

- $k = 0$. Then σ^S_0 is both defined and undefined at (U,P), a contradiction.
- $k > 0$. There are four cases for the event label.
  - λ^S_{k-1} is of the form $\mathsf{snd}\langle \widehat{Rcvr}, \widehat{U}, \widehat{P}, \widehat{\Pi_{\mathrm{pl}}}, \widehat{\Pi_{\mathrm{pr}}}\rangle$ for some $\widehat{Rcvr}$, $\widehat{U}$, $\widehat{P}$, $\widehat{\Pi_{\mathrm{pl}}}$, $\widehat{\Pi_{\mathrm{pr}}}$. Then $\sigma^S_{k-1}(U,P) = (\Pi_{\mathrm{pl}}, \Pi_{\mathrm{pr}})$. Apply the induction hypothesis. Then notice that the receiving position before k is the same as receiving position before $k-1$.
  - λ^S_{k-1} is of the form $\mathsf{rcv}\langle \widehat{Sndr}, \widehat{U}, \widehat{P}, \widehat{\Pi_{\mathrm{pl}}}, \widehat{\Pi_{\mathrm{pr}}}\rangle$ for some principal identifier $\widehat{Sndr}$, user identifier $\widehat{U}$, PII category $\widehat{P}$, $(\widehat{U}, S)$-policy $\widehat{\Pi_{\mathrm{pl}}}$ that satisfies the $(\widehat{U}, S)$-preference $\widehat{\Pi_{\mathrm{pr}}}$ and $\sigma^S_k = \sigma^S_{k-1}[(\widehat{U}, \widehat{P}) \mapsto (\widehat{\Pi_{\mathrm{pl}}}, \widehat{\Pi_{\mathrm{pr}}})]$. We have two cases.
    - $(U,P) \neq (\widehat{U}, \widehat{P})$. Apply the induction hypothesis. Notice that the receiving position of P for U before k is the same as before $k-1$.
    - $(U,P) = (\widehat{U}, \widehat{P})$. Then $\widehat{\Pi_{\mathrm{pl}}} = \Pi_{\mathrm{pl}}$ and $\widehat{\Pi_{\mathrm{pr}}} = \Pi_{\mathrm{pr}}$ and the receiving position of P for U before k is $r = k-1$. In particular, Π_{pl} satisfies Π_{pr}. By Thm. 4.8, if $\mathrm{trace}_r(\tilde{R}(S))$ complies with

$[\![\Pi_{\text{pl}}]\!]^{\text{pl}}_{\tau,\mathcal{A}_{\text{pr}}}$, it also complies with $[\![\Pi_{\text{pr}}]\!]^{\text{pr}}_{\tau,\mathcal{A}_{\text{pl}}}$. There are two possibilities for $\widehat{\mathit{Sndr}}$.

- $\widehat{\mathit{Sndr}} \in \mathbf{Usr}$. By definition of an evolving run with transitive disclosures we have $\widehat{\mathit{Sndr}} = \widehat{U}$, so $\widehat{\mathit{Sndr}} = U$. By definition of a synchronized run, $\lambda^{U}_{k-1} = \mathsf{snd}\langle S, U, P, \Pi_{\text{pl}}, \Pi_{\text{pr}}\rangle$.
- $\widehat{\mathit{Sndr}} \in \mathbf{Svc}$. Let $\mathit{Sndr} = \widehat{\mathit{Sndr}}$. By definition of a synchronized run, $\lambda^{\mathit{Sndr}}_{k-1} = \mathsf{snd}\langle S, U, P, \Pi_{\text{pl}}, \Pi_{\text{pr}}\rangle$. Let $\tau' = (U, \mathit{Sndr})$. By definition of an evolving run with transitive disclosures there is a τ'-policy $\Pi'_{\text{pl}} = (\mathcal{A}'_{\text{pl}}, q'_{\text{m}})$ and a τ'-preference $\Pi'_{\text{pr}} = (\mathcal{A}'_{\text{pr}}, q'_{\text{w}})$ such that $\sigma^{\mathit{Sndr}}_{k-1}(U, P) = (\Pi'_{\text{pl}}, \Pi'_{\text{pr}})$ and $\langle \texttt{send}\ P\ \texttt{to}\ S\rangle \in \mathbf{Beh}(t')$ where $t' = \text{trace}_{k-1}(\tilde{R}(\mathit{Sndr}))$. By induction hypothesis, Π'_{pl} satisfies Π'_{pr}. Assume that t' complies with $[\![\Pi'_{\text{pl}}]\!]^{\text{pl}}_{\tau',\mathcal{A}'_{\text{pr}}}$. By Thm. 4.8, t' complies with $[\![\Pi'_{\text{pr}}]\!]^{\text{pr}}_{\tau',\mathcal{A}'_{\text{pl}}}$. By Def. 4.4, $\mathbf{Beh}(t') \models^{\text{ma}}_{\tau',\mathcal{A}'_{\text{pl}}} \mathcal{A}'_{\text{pr}}$. By definition of permissions allowed by preferences, $\mathbf{Beh}(t') \subseteq \{B \mid \mathcal{A}'_{\text{pr}} \cup \mathcal{A}'_{\text{pl}} \vdash U\ \mathsf{says}\ \mathit{Sndr}\ \mathsf{may}\ B\}$. Thus $\{\langle \texttt{send}\ P\ \texttt{to}\ S\rangle\} \subseteq \{B \mid \mathcal{A}'_{\text{pr}} \cup \mathcal{A}'_{\text{pl}} \vdash U\ \mathsf{says}\ \mathit{Sndr}\ \mathsf{may}\ B\}$. So $\{\langle \texttt{send}\ P\ \texttt{to}\ S\rangle\} \models^{\text{ma}}_{\tau',\mathcal{A}'_{\text{pl}}} \mathcal{A}'_{\text{pr}}$.

– λ^{S}_{k-1} is of the form $\text{chg}\langle \widehat{U}, \widehat{P}, \widehat{\Pi^{\text{new}}_{\text{pl}}}\rangle$ for some $\widehat{U} \in \mathbf{Usr}$, $\widehat{P} \in \mathbf{PII}$ and a $(\widehat{U}, S)$-policy $\widehat{\Pi^{\text{new}}_{\text{pl}}}$. There are two cases.
  - $*$ $(U, P) \neq (\widehat{U}, \widehat{P})$. Then $\sigma^{S}_{k-1}(U, P) = (\Pi_{\text{pl}}, \Pi_{\text{pr}})$. Apply the induction hypothesis. Notice that the receiving position of P for U before k is the same as before $k-1$.
  - $*$ $(U, P) = (\widehat{U}, \widehat{P})$. By definition of an evolving run with transitive disclosures, $\Pi_{\text{pl}} = \widehat{\Pi^{\text{new}}_{\text{pl}}}$ and there is a (U, S)-policy $\widehat{\Pi^{\text{old}}_{\text{pl}}}$ and a (U, S)-preference Π_{pr} such that $\sigma^{S}_{k-1}(U, P) = (\widehat{\Pi^{\text{old}}_{\text{pl}}}, \Pi_{\text{pr}})$ and $\widehat{\Pi^{\text{new}}_{\text{pl}}}$ satisfies Π_{pr}. Then Π_{pl} satisfies Π_{pr}. By Thm. 4.8, if $\text{trace}_r(\tilde{R}(S))$ complies with $[\![\Pi_{\text{pl}}]\!]^{\text{pl}}_{\tau,\mathcal{A}_{\text{pr}}}$, it also complies with $[\![\Pi_{\text{pr}}]\!]^{\text{pr}}_{\tau,\mathcal{A}_{\text{pl}}}$. Apply the induction assumption to $k-1$ and $\sigma^{S}_{k-1}(U, P) = (\widehat{\Pi^{\text{old}}_{\text{pl}}}, \Pi_{\text{pr}})$ to obtain $\Pi^{\text{orig}}_{\text{pl}}$ and the receiving position r of P for U before $k-1$. Notice that r is also the receiving position of P for U before k.

– $\lambda^{S}_{k-1} = \epsilon$. Then $\sigma^{S}_{k-1} = \sigma^{S}_{k}$, apply the induction hypothesis and notice that the receiving position of P for U before $k-1$ is the same as the receiving position of P for U before k.

□

6 Discussion

Privacy vs. access control. S4P's syntax is based on SecPAL [8], a language for specifying access control policies in decentralized systems. We chose to adopt the syntax of an access control language because of the close links between privacy management and access control. In access control, a service specifies an access control policy consisting of assertions that govern who can access which resources, and user requests trigger queries against this policy. In privacy management, the roles of permission granter and seeker are reversed. It is now the user who wishes to protect her data, but we can reuse the same mechanisms as in access control: the user specifies a privacy preference consisting of assertions that govern which services may use her data, and disclosure requests trigger queries against the preference assertions.

Privacy management is not only concerned with permissions but also with promises and obligations. However, we realized that promises can be viewed as dual to permissions, and can be expressed as assertions on the service's side, and queries on the user's side. Consequently, we can reuse the query evaluation mechanism from the access control language to check satisfaction between privacy policies and preferences.

From the rather large family of existing access control languages (see Section 2), we chose SecPAL because its syntax is close to natural language and its design is very abstract. Apart from the special can say construct, it does not commit to any vocabulary, hence it can be instantiated to serve a wide range of different purposes. In particular, it lets us introduce the may and will modalities to express permissions and promises in privacy management. Additionally, SecPAL provides the basic infrastructure for specifying highly expressive policies: parameterized predicates, if-clauses, arbitrary constraints, and fine-grained delegation of authority.

Implementation. Our implementation of S4P is based on the SecPAL [8] evaluation engine, extended with generic predicates and the may/will-constructs. The evaluation process begins by translating each assertion into one or more constrained Datalog [23] clauses. Queries against the resulting constrained Datalog program are evaluated using a resolution algorithm with tabling [18], in order to guarantee termination even with recursive policies and preferences. The translation preserves S4P's query semantics in the sense that a query is true in the context of a S4P policy or preference iff the corresponding Datalog query evaluates to true against the Datalog program.

We also developed a number of tools that help debug policies and preferences. The result of a successful query can be visualized by a proof viewer that graphically displays the deduction steps in the form of a proof graph. Failed queries can be analyzed using our tool based on logical abduction [9], which computes a set of missing S4P facts that would have satisfied the query.

The syntax of our language has been designed to make preferences and policies as human-readable as possible. However, while professional hosting services may be able to directly write their policies in S4P using a dedicated editor with tool support, end users cannot be expected to be willing to learn the language, so where would they get their preferences from? First of all, users are offered to select amongst a small number of predefined default preferences for specific types of services. Preferences could be customized using application-specific or browser-specific user interfaces that do not offer the full expressiveness and flexibility of the underlying language, but let the user extend or define exceptions to the predefined preferences. User agents may also be able to download default preferences provided by trusted third parties for specific application domains. Furthermore, the task of managing preferences and of checking satisfaction between policies and preferences could be delegated to an external adviser acting as a user agent. We are working on a method based on the abductive tool mentioned above to make suggestions to the user on how her preference could be modified, in the case of a mismatch, in order for a service policy to satisfy it.

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 [16, 22].

We have found that for typical policies, queries are evaluated within a few milliseconds. More expensive queries could be constructed in principle, but performance is not a vital issue in this context, as the satisfaction check is usually not performed by the (possibly congested) service, but by the user. Moreover, privacy policies and preferences tend to be small and simple compared to authorization policies (which may typically contain millions of facts).

Expressiveness. We now discuss frequent features of natural language privacy policies for which the mapping to S4P is not completely straightforward.

Many policies allow users to access their own data. For instance, Facebook's privacy policy[3] states that "users may modify or delete any of their profile information at any time by logging into their account". One might think that in order to encode such a statement in S4P, the service policy would need to contain may-assertions. (Recall that only preferences contain may-assertions, while policies contain may-queries.) However, the purpose of such statements is not to express an upper bound on the user's behaviours, but to *promise* to provide the user access to her data. Hence such statements should be expressed as will-assertions, just like other obligations:

```
FB says FB will allow ⟨Usr⟩ to
    delete data of type ProfileInfo
```

Many services allow the user to opt in or opt out of specific service behaviours. Ebay.com is a prime example for this feature, providing a web page on which users can configure more than 50 options regarding email notifications. Other services specify the options in the natural language privacy policy itself. We have chosen not to model user options directly in the language. Instead, external mechanisms can be used to negotiate a policy and a preference between the service and the user. For example, ticking an opt-in checkbox on a web page provided by the service and displayed by the user agent could automatically add a conjunction to the service's may-query and a may-assertion to the user's preference.

[3]Specific privacy policies mentioned in this section refer to versions retrieved from respective public websites on 01/02/2010.

Real-world preferences and policies usually contain a large number of assertions, for instance if there are many different types of PIIs for which preferences and policies have to be specified. We dealt with this complexity by implementing *hierarchical* predicate parameters, and to specify PII types (and possibly other types such as usage purposes) to be hierarchical. Under this scheme, the fact `Alice` says ⟨Svc⟩ may use `/Addr` for p automatically implies that ⟨Svc⟩ can also use `/Addr/Email`, `/Addr/Email/Secondary`, `/Addr/Postcode` etc.

However, combining hierarchical types with may-queries and will-queries can lead to subtle results. For example, the will-query

```
Alice says ⟨Svc⟩ will delete /Addr within 30 days?
```

is not satisfied by the will-assertion

```
eBooking says eBooking
  will delete /Addr/Email within 30 days
```

even if the PII to be disclosed is the email address. For the query to be satisfied, the service actually has to promise to delete `/Addr` or / (i.e., the top-level PII type).

An alternative solution (which can be combined with hierarchical parameters) would be to introduce a placeholder ⟨PII⟩ that gets instantiated before the satisfaction check with the PII type in question, and a simple syntactic scoping construct that takes advantage of the hierarchy on PII types:

```
Include if ⟨PII⟩ ⪯ /Addr {
  [...]
  Alice says ⟨Svc⟩ will delete ⟨PII⟩ within 30 days?
  [...]
}
```

The importance of being abstract. In designing S4P, we have deliberately tried to keep the language as abstract as possible, while still being able to express the essence of privacy policies and preferences, namely upper and lower bounds on service behaviours. Abstractness has many advantages. It 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 more modular language design, which also simplifies formal reasoning. As we have seen, a number of 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 of all, we have kept the vocabulary abstract. Even though most websites' natural language privacy statements have a similar structure (more or less adhering to the EU-US Safe Harbor Privacy Principles[4]), with details on notification, user choice, third party disclosure,

[4]`http://www.export.gov/safeharbor/`

user access, and security measures, their choice of vocabulary varies greatly, especially across different application domain. For example, travel booking websites use verb phrases such as "make travel arrangements on your behalf" and constants such as "frequent flyer membership number" (from Expedia.co.uk), whereas a webmail service may require behaviours such as "preventing unsolicited bulk email" and constants such as "contact list" (from Gmail.com). But vocabularies also differ within the same application domain. For example, the Microsoft HealthVault policy refers to people with delegated access rights to health data as "custodians", whereas the corresponding term in the Google Health privacy policy is "people you trust".

Secondly, 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 in the case of behaviours that involve 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. Part 5.2 presents an example of how the abstract behaviour mapping can be partially concretized: the definition of an evolving run with transitive disclosures gives semantics to the $\langle \mathsf{send}\ \_\ \mathsf{to}\ \_\rangle$ behaviour, and stating and proving a correctness property about just this specific behaviour is relatively easy because all other behaviours have been kept abstract. The works by Ni *et al.* [28] on modelling complex privacy obligations, and by Barth *et al.* [5] on using temporal logic to express trace constraints, are examples of how our abstract notion of behaviour could be concretized.

Finally, we are not tied to a specific compliance enforcement model. In practice, it is often unfeasible and unnecessary to automatically enforce compliance; instead, informal methods such as auditing are used. To automate enforcement, the most straightforward solution is to implement a reference monitor for dynamically checking the permissions, accompanied by an obligation monitoring system (e.g. [11, 21, 26]). For simple systems, it may be possible to enforce compliance by static analysis, as opposed to dynamic monitoring, as has been done for cryptographic protocols and access control policies [10]. We plan to investigate this direction in future work.

In conclusion, 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] W. Adkinson, J. Eisenach, and T. Lenard. Privacy online: a report on the information practices and policies of commercial web sites. Progress and Freedom Foundation, Mar 2002.

[2] A. Antón, J. Earp, D. Bolchini, Q. He, C. Jensen, W. Stufflebeam, et al. The lack of clarity in financial privacy policies and the need for standardization. In *IEEE Symposium on Security & Privacy*, pages 36–45, 2004.

[3] 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.

[4] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise Privacy Authorization Language (EPAL 1.2). Technical report, IBM, Nov. 2003.

[5] A. Barth, A. Datta, J. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *IEEE Symposium on Security and Privacy*, pages 184–198. IEEE Computer Society, 2006.

[6] 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.

[7] 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.

[8] M. Y. Becker, C. Fournet, and A. D. Gordon. Design and semantics of a decentralized authorization language. In *IEEE Computer Security Foundations Symposium*, pages 3–15, 2007.

[9] M. Y. Becker and S. Nanz. The role of abduction in declarative authorization policies. In *10th International Symposium on Practical Aspects of Declarative Languages (PADL)*, 2008.

[10] J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement types for secure implementations. In *21st IEEE Computer Security Foundations Symposium*, pages 17–32, 2008.

[11] C. Bettini, S. Jajodia, X. Wang, and D. Wijesekera. Obligation monitoring in policy management. In *Policies for Distributed Systems and Networks*, 2002.

[12] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[13] 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.

[14] 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. http://www.w3.org/TR/P3P11.

[15] L. Cranor, M. Langheinrich, and M. Marchiori. *A P3P Preference Exchange Language 1.0*. W3C, Apr. 2002. `http://www.w3.org/TR/P3P-preferences`.

[16] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. In *CCC '97: Proceedings of the 12th Annual IEEE Conference on Computational Complexity*, 1997.

[17] E. Denham. Report of findings into the complaint filed by the Canadian Internet Policy and Public Interest Clinic (CIPPIC) against Facebook Inc. Office of the Privacy Commissioner of Canada, 2009. `http://priv.gc.ca/cf-dc/2009/2009_008_0716_e.pdf`.

[18] S. W. Dietrich. Extension tables: Memo relations in logic programming. In *Symposium on Logic Programming*, pages 264–272, 1987.

[19] Federal Trade Commission. Sony BMG Music settles charges its music fan websites violated the Childrens Online Privacy Protection Act, Dec 2008. `http://www.ftc.gov/opa/2008/12/sonymusic.shtm`.

[20] 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.

[21] K. Irwin, T. Yu, and W. H. Winsborough. On the modeling and analysis of obligations. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 134–143, 2006.

[22] A. Itai and J. A. Makowsky. Unification as a complexity measure for logic programming. *Journal of Logic Programming*, 4(2), 1987.

[23] J. Jaffar and M. J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[24] C. Jensen and C. Potts. Privacy policies as decision-making tools: an evaluation of online privacy notices. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 471–478, New York, NY, USA, 2004. ACM.

[25] C. M. Kelley, A. Denton, and R. Broadbent. Privacy concerns cost eCommerce $15 billion. Forrester Research, Sep 2001.

[26] K. Krukow, M. Nielsen, and V. Sassone. A logical framework for history-based access control and reputation systems. *Journal of Computer Security*, 16(1):63–101, 2008.

[27] D. Métayer. A formal privacy management framework. In *Formal Aspects in Security and Trust*, pages 162–176, 2009.

[28] Q. Ni, E. Bertino, and J. Lobo. An obligation model bridging access control

policies and privacy policies. In *ACM symposium on access control models and technologies*, pages 133–142, 2008.

[29] OASIS. *eXtensible Access Control Markup Language (XACML) Version 2.0 core specification*, 2005. At `www.oasis-open.org/committees/xacml/`.

[30] W. H. Stufflebeam, A. I. Antón, Q. He, and N. Jain. Specifying privacy policies with P3P and EPAL: lessons learned. In *ACM workshop on privacy in the electronic society*, pages 35–35, 2004.

[31] Superior Court of California, County of Orange. Elisha Melkonian et al. vs. Facebook Inc., No. 30-2009, 2009.

[32] United States District Cout for the Southern District of New York. Memorandum of points and authorities in support of EPIC's motion to intervene. Case No. 05 CV 8136-DC, 2009. `http://epic.org/privacy/googlebooks/EPIC_Brief-GBS.pdf`.

A Extended Example

Alice's privacy preference. Alice cares about online child protection, so her privacy preference for web content providers contains the following will-query:

$$\langle\mathsf{Svc}\rangle \text{ says } \langle\mathsf{Svc}\rangle \text{ will allow } \langle\mathsf{Usr}\rangle \text{ to } \texttt{Edit} \text{ ParentalControls? } \wedge \\ \texttt{Alice} \text{ says } \langle\mathsf{Svc}\rangle \text{ complies with } \texttt{COPPA}? \tag{1}$$

According to this will-query, Alice requires web content 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. But she does not know the entire list of such programs either, so she delegates authority over such schemes to the FTC. She also has a statement from the FTC saying that TRUSTe is such a scheme.

$$\texttt{Alice} \text{ says } x \text{ can say } y \text{ complies with } \texttt{COPPA} \text{ if} \\ \quad x \text{ is member of } \texttt{COPPACompliancySchemes} \tag{2}$$

$$\texttt{Alice} \text{ says } \texttt{FTC} \text{ can say } x \text{ is member of } \texttt{COPPACompliancySchemes} \tag{3}$$

$$\texttt{FTC} \text{ says } \texttt{TRUSTe} \text{ is member of } \texttt{COPPACompliancySchemes} \tag{4}$$

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. The last two assertions are default statements that allow service behaviours that should obviously be allowed. A more realistic privacy preference would of course include many more may-assertions.

$$\langle\mathsf{Usr}\rangle \text{ says } \langle\mathsf{Svc}\rangle \text{ may use } \texttt{Cookies} \text{ for } x \text{ if} \\ \quad \langle\mathsf{Svc}\rangle \text{ will revoke } \texttt{Cookies} \text{ within } t \\ \quad \text{where } t \leq \texttt{5yr} \tag{5}$$

$$\langle\mathsf{Usr}\rangle \text{ says } \langle\mathsf{Svc}\rangle \text{ can say } \langle\mathsf{Svc}\rangle \text{ will revoke } \texttt{Cookies} \text{ within } t \tag{6}$$

$$\texttt{Alice} \text{ says } \langle\mathsf{Svc}\rangle \text{ may allow } \texttt{Alice} \text{ to } \textit{action object} \tag{7}$$

$$\texttt{Alice} \text{ says } \langle\mathsf{Svc}\rangle \text{ may revoke } \texttt{Cookies} \text{ within } t \tag{8}$$

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

$$\texttt{Alice} \text{ says } \texttt{Alice} \text{ is using software } \texttt{MSNClient} \text{ version } 9.5 \tag{9}$$

Microsoft's privacy policy. The English statements in *italics* are taken verbatim from Microsoft's Online Privacy Statement[5].

Microsoft is a member of the TRUSTe Privacy Programme. This means that Microsoft complies with a number of privacy standards including, in particular, COPPA.

$$\texttt{TRUSTe}\ \mathsf{says}\ \texttt{MS}\ \mathsf{complies\ with}\ \texttt{COPPA} \qquad (10)$$

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.

$$\begin{array}{l} \texttt{MS}\ \mathsf{says}\ \texttt{MS}\ \mathsf{will\ allow}\ \langle\mathsf{Usr}\rangle\ \mathsf{to}\ \texttt{Edit}\ \mathsf{ParentalControls\ if} \\ \quad \langle\mathsf{Usr}\rangle\ \mathsf{is\ member\ of}\ \mathit{msntype}, \\ \quad \mathit{msntype}\ \mathsf{supports\ parental\ controls}, \\ \quad \langle\mathsf{Usr}\rangle\ \mathsf{is\ using\ software}\ \texttt{MSNClient}\ \mathsf{version}\ v \\ \quad \mathsf{where}\ v \leq 9.5 \end{array} \qquad (11)$$

$$\texttt{MS}\ \mathsf{says}\ \texttt{MSNPremium}\ \mathsf{supports\ parental\ controls} \qquad (12)$$

$$\texttt{MS}\ \mathsf{says}\ \texttt{MNSPlus}\ \mathsf{supports\ parental\ controls} \qquad (13)$$

$$\texttt{MS}\ \mathsf{says}\ \texttt{MSN9DialUp}\ \mathsf{supports\ parental\ controls} \qquad (14)$$

The various types of MSN membership are delegated to MSN:

$$\texttt{MS}\ \mathsf{says}\ \texttt{MSN}\ \mathsf{can\ say}\ x\ \mathsf{is\ member\ of}\ \texttt{MSN} \qquad (15)$$

$$\texttt{MS}\ \mathsf{says}\ \texttt{MSN}\ \mathsf{can\ say}\ x\ \mathsf{is\ member\ of}\ \texttt{MSNPremium} \qquad (16)$$

$$\texttt{MS}\ \mathsf{says}\ \texttt{MSN}\ \mathsf{can\ say}\ x\ \mathsf{is\ member\ of}\ \texttt{MSNPlus} \qquad (17)$$

$$\texttt{MS}\ \mathsf{says}\ \texttt{MSN}\ \mathsf{can\ say}\ x\ \mathsf{is\ member\ of}\ \texttt{MSN9DialUp} \qquad (18)$$

In particular, MSN knows that Alice has a MSNPremium account. In our implementation, such assertions can be created on the fly and fetched during evaluation using interfaces to database management systems and directory services such as SQL Server and Active Directory.

$$\texttt{MSN}\ \mathsf{says}\ \texttt{Alice}\ \mathsf{is\ member\ of}\ \texttt{MSNPremium} \qquad (19)$$

MSN trusts users on the version of their software:

$$\texttt{MS}\ \mathsf{says}\ \langle\mathsf{Usr}\rangle\ \mathsf{can\ say}\ \langle\mathsf{Usr}\rangle\ \mathsf{is\ using\ software}\ \texttt{MSNClient}\ \mathsf{version}\ v\,. \qquad (20)$$

[5]Retrieved from http://privacy.microsoft.com/en-gb/fullnotice.mspx, version from July 2009.

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.*

$$\langle \mathsf{Usr} \rangle \text{ says } \mathtt{MS} \text{ may use } \mathtt{Cookies} \text{ for } \mathtt{AdTracking}? \tag{21}$$

The cookies we use for advertising have an expiry date of no more than 2 years.

$$\mathtt{MS} \text{ says } \mathtt{MS} \text{ will revoke } \mathtt{Cookies} \text{ within } \mathtt{2yr} \tag{22}$$

The may-query above is conjoined with a number of trivial statements declaring all relevant behaviours for this encounter:

$$\begin{aligned}&\langle \mathsf{Usr} \rangle \text{ says } \mathtt{MS} \text{ may revoke } \mathtt{Cookies} \text{ within } \mathtt{2yr}? \wedge \\ &\langle \mathsf{Usr} \rangle \text{ says } \mathtt{MS} \text{ may allow } \langle \mathsf{Usr} \rangle \text{ to } \mathtt{Edit} \text{ ParentalControls}? \end{aligned} \tag{23}$$

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 (19) which has been delegated authority over MSN Premium memberships (16). Furthermore, MSN Premium accounts support parental controls according to (12), Alice is using a version of MSN client that supports parental controls (9), and is trusted on this fact (20).

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 (21,23) also consists of multiple conjuncts. The first one (21) is satisfied by Alice's assertion (5) which in turn depends on (6) and Microsoft's will-assertion (22). The remaining two conjuncts (23) are satisfied by Alice's may-assertions (7) and (8).

Hence Alice's preference is satisfied by the policy.

B Proofs

Restatement of Lemma 4.5. Let $\mathcal{A}$ be a set of assertions, q_m a closed τ-may-query, and $\mathcal{B}$ a set of ground behaviour atoms. If $\mathcal{A} \vdash q_m$ and $\mathcal{B} \models^{\mathrm{mq}}_{\tau,\mathcal{A}} q_m$ then $\mathcal{B} \models^{\mathrm{ma}}_{\tau,\mathcal{A}} \mathcal{A}$.

Proof. By structural induction on q_m. We assume (a) $\mathcal{A} \vdash q_m$ and (b) $\mathcal{B} \models^{\mathrm{mq}}_{\tau,\mathcal{A}} q_m$. It is sufficient to show that $\mathcal{B} \subseteq \mathcal{B}' = \{B \mid \mathcal{A} \vdash U \text{ says } S \text{ may } B\}$. There are three cases to consider.

In the first case, q_m is of the form U $\textsf{says}$ S $\textsf{may}$ B?. From (a), $B \in \mathcal{B}'$. From (b), $\mathcal{B} \subseteq \{B\}$. Hence $\mathcal{B} \subseteq \mathcal{B}'$.

In the second case, q_m is of the form $q_1 \wedge q_2$. From (b) and the induction hypothesis, either there exist $\mathcal{B}_1$ and $\mathcal{B}_2$ such that $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2$ and $\mathcal{B}_1 \subseteq \mathcal{B}'$ and $\mathcal{B}_2 \subseteq \mathcal{B}'$; therefore $\mathcal{B} \subseteq \mathcal{B}'$. Or else this case is subsumed by the last and third case below.

In the third case, no subquery of the form U $\textsf{says}$ S $\textsf{may}$ B? occurs in q_m. From (b), $\mathcal{B} = \emptyset$, hence trivially $\mathcal{B} \subseteq \mathcal{B}'$. □

Restatement of Lemma 4.6. Let $\mathcal{A}$ be a set of assertions, q_w a closed τ-will-query, and $\mathcal{B}$ a set of ground behaviour atoms. If $\mathcal{A} \vdash q_w$ and $\mathcal{B} \models^{\mathrm{wa}}_{\tau,\mathcal{A}} \mathcal{A}$ then $\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_w$.

Proof. By structural induction on q_w. We assume (a) $\mathcal{A} \vdash q_w$ and (b) $\mathcal{B} \supseteq \mathcal{B}' = \{B \mid \mathcal{A} \vdash S \text{ says } S \text{ will } B\}$.

Consider the case where q_w is of the form S $\textsf{says}$ S $\textsf{will}$ B?. From (a), $B \in \mathcal{B}'$. Together with (b), this gives $\mathcal{B} \supseteq \{B\}$, and hence $\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_w$. The other cases follow from (a) and the induction hypothesis. □

Restatement of Lemma 4.7. Let $\Pi_{\mathrm{pl}} = (\mathcal{A}_{pl}, q_m)$ be a τ-policy and $\Pi_{\mathrm{pr}} = (\mathcal{A}_{pr}, q_w)$ a τ-preference. If Π_{pl} satisfies Π_{pr} then $[\![\Pi_{\mathrm{pl}}]\!]^{\mathrm{pl}}_{\tau,\mathcal{A}_{pr}}$ is at least as strict as $[\![\Pi_{\mathrm{pr}}]\!]^{\mathrm{pr}}_{\tau,\mathcal{A}_{pl}}$.

Proof. Suppose $t \in [\![\Pi_{\mathrm{pl}}]\!]^{\mathrm{pl}}_{\tau,\mathcal{A}_{pr}}$. Let $\mathcal{B} = \mathbf{Beh}(t)$. We need to show that $t \in [\![\Pi_{\mathrm{pr}}]\!]^{\mathrm{pr}}_{\tau,\mathcal{A}_{pl}}$, that is, (a) $\mathcal{B} \models^{\mathrm{ma}}_{\tau,\mathcal{A}_{pl}} \mathcal{A}_{pr}$ and (b) $\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}_{pl}\cup\mathcal{A}_{pr}} q_w$.

From the assumption, $\mathcal{B} \models^{\mathrm{mq}}_{\tau,\mathcal{A}_{pl}\cup\mathcal{A}_{pr}} q_m$. From the definition of satisfaction and Lemma 4.5, $\mathcal{B} \models^{\mathrm{ma}}_{\tau,\mathcal{A}_{pl}\cup\mathcal{A}_{pr}} \mathcal{A}_{pl} \cup \mathcal{A}_{pr}$. By definition of $\models^{\mathrm{ma}}_{\tau,\mathcal{A}_{pl}\cup\mathcal{A}_{pr}}$, we also have (a) $\mathcal{B} \models^{\mathrm{ma}}_{\tau,\mathcal{A}_{pl}} \mathcal{A}_{pr}$.

From the definition of satisfaction we get $\mathcal{A}_{pl} \cup \mathcal{A}_{pr} \vdash q_w$. Furthermore, from the assumption we get $\mathcal{B} \models^{\mathrm{wa}}_{\tau,\mathcal{A}_{pr}} \mathcal{A}_{pl}$, which is equivalent to $\mathcal{B} \models^{\mathrm{wa}}_{\tau,\mathcal{A}_{pl}\cup\mathcal{A}_{pr}} \mathcal{A}_{pl} \cup \mathcal{A}_{pr}$ by definition of $\models^{\mathrm{wa}}_{\tau,\mathcal{A}_{pl}\cup\mathcal{A}_{pr}}$. Hence Lemma 4.6 can be applied to get (b) $\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}_{pl}\cup\mathcal{A}_{pr}} q_w$. □

Restatement of Theorem 4.8. Let $\Pi_{\mathrm{pl}} = (\mathcal{A}_{pl}, q_m)$ be a τ-policy and $\Pi_{\mathrm{pr}} = (\mathcal{A}_{pr}, q_w)$ a τ-preference. If a trace t complies with $[\![\Pi_{\mathrm{pl}}]\!]^{\mathrm{pl}}_{\tau,\mathcal{A}_{pr}}$ and Π_{pl} satisfies Π_{pr}, then t complies with $[\![\Pi_{\mathrm{pr}}]\!]^{\mathrm{pr}}_{\tau,\mathcal{A}_{pl}}$.

Proof. This is a corollary of Lemma 4.7 and Lemma 4.2. □