A Framework for Privacy Preferences and Data-Handling Policies

Moritz Y. Becker Alexander Malkis
Laurent Bussard

September 2009

Technical Report
MSR–TR–2009–128

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

A Framework for Privacy Preferences and Data-Handling Policies

Moritz Y. Becker Alexander Malkis Laurent Bussard

September 2009

This paper presents SecPAL4P, a language for specifying both users' preferences on how their personally identifiable information (PII) should be treated by data-collecting services, and services' policies on treating collected PIIs. Preferences and policies are specified in terms of granted rights and required obligations, expressed as assertions and queries in an instance of SecPAL (a language originally developed for decentralized authorization). This paper further presents a formal definition of satisfaction between a policy and a preference, and a satisfaction checking algorithm. Based on the latter, a protocol is described for disclosing PIIs between users and services, as well as between third-party services.

1 Introduction

Increasing amounts of personal information are being collected by Internet services today. For instance, commercial sites typically require users to provide their credit card details and contact information. In social networks, users voluntarily disclose personal information to large communities. In cloud computing, users' applications and data reside in the "cloud", spanning multiple regulatory environments. Finally, the community-wide electronic health record services that are being developed around the globe will store a vast amount of patient-confidential data.

Naturally, users are becoming increasingly aware and concerned about privacy issues, and want to disclose their personal information only to services that they trust to handle their data in a way that satisfies their (usually implicit) preferences. However, most users do not want to invest much time in reading websites' privacy policies or in configuring their own preferences. Indeed, in most domains, support for privacy is not yet seen as a competitive advantage, and pressure from users and regulators (see e.g. European directives on privacy [13, 14]) is necessary to convince service providers.

Currently, there is no satisfactory solution for specifying and matching users' privacy preferences and services' data-handling policies. Languages such as P3P [20], APPEL [19], EPAL[1], and Prime-DHP [3] lack a formal specification of when a policy satisfies a preference, are not easily human-readable, and cannot

express certain statements that seem useful in practice. The latter include obligations (requirements on the service's behaviour, e.g. to forget the data after some maximum period of time) and restrictions on third-party information disclosure, where the user's data may be forwarded between multiple services.

There are two main issues to be considered when personal data is about to be disclosed and collected by a service. Firstly, the user and the service have to agree on how the data will be handled by the service. In other words, it has to be checked whether the service's data-handling policy satisfies the user's privacy preference on this piece of data. Secondly, services have to be trusted to comply with their own data-handling policies.

This work solves the first issue by offering a mechanism for specifying user preferences and service policies, and for checking satisfaction between policies and preferences. The second issue is partially addressed by providing a framework that formally specifies the notion of compliance. This framework is deliberately kept abstract; concrete mechanisms for checking or enforcing compliance are therefore not within the scope of this work.

Our main technical contributions are as follows:

1. An abstract framework that formalises the notions of preferences and policies, as well as satisfaction and compliance.

2. A highly expressive, declarative language for expressing both users' privacy preferences and services' data-handling policies. The language supports permissions and obligations, is close to natural language, and has a formal semantics. The language is based on SecPAL, a language originally designed for specifying authorization policies in decentralized systems. We call our language *SecPAL for Privacy* (SecPAL4P).

3. A data disclosure protocol for determining if a service's policy satisfies a user's preference, and ensuring that the preference is respected when data is forwarded to and between third party services.

This paper is organized as follows: Section 2 gives an informal overview of SecPAL4P and briefly summarizes the underlying language SecPAL. Section 3 illustrates the language and the protocol by means of an extended example. Section 4 defines SecPAL4P as an instance of SecPAL, and presents an algorithm for checking satisfaction between a policy and a preference. Section 5 presents the abstract framework that formalises the notions of preference, policy, satisfaction and compliance. Section 6 describes the protocol for data disclosure and fowarding. Related work is discussed in Section 7. We conclude in Section 8 with a discussion on possible extensions, limitations of our approach, and potential future work. Finally, proofs are provided in an appendix.

2 Overview

The user has a collection of pieces of personally identifiable information (PII). The user specifies a *preference* on handling her PIIs. Services specify a data-

handling *policy* on treating users' PIIs. During an *encounter* between a user and a service, it is checked whether the service's policy *satisfies* the user's preference. If a service *complies* with its own policy, then this check guarantees that it also complies with the user's preference. Policies and preferences may also specify the conditions when a PII may (or must) be *forwarded* to a third-party service. A protocol for data forwarding should ensure that a service may only get hold of a user's PII if this particular communication of the PII is in accordance with the user's preference, and the service's policy satisfies the preference. The framework presented here provides a language for specifying user preferences and service policies, and a method for checking satisfaction between a policy and a preference.

The following gives a more detailed intuition on the intended meaning of preferences and policies. We assume that there is a predefined collection of PII-relevant service *behaviours*, and a corresponding vocabulary for representing these behaviours. These are generally domain-specific, and may include "using an email address for marketing", "forwarding contact details to trusted sellers", "deleting credit card details within one month" or "retaining X-rays for at least 10 years".

A user preference can be divided into two parts. The first part specifies an *upper bound* on a service's behaviours with respect to the user's PIIs. It therefore expresses what a service is *permitted* to do. The second part specifies a *lower bound* on a service's behaviours. It therefore expresses *obligations*, i.e. the behaviours that a service must exhibit towards a PII.

A service policy can also be divided into two parts. The first part specifies an *upper bound* on its own PII-relevant behaviours. It therefore expresses and advertises the *possible* behaviours of the service. The second part specifies a *lower bound* on its behaviours. Therefore, these are *promised* behaviours.

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. 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.

This duality is reflected in the language. The upper bound on behaviours is specified as phrases using the may verb. More specifically, the upper bound is specified in the user preference as a collection of may-*assertions*, e.g.

⟨Usr⟩ says ⟨Svc⟩ may use FaxNo for Contact.

In the service policy, the upper bound is specified as a may-*query*, because the corresponding possible behaviours should be a subset of the permitted behaviours. Intuitively, a service must ask for permission upfront for anything that it might do with a user's PII in the future. Here is a simple may-*query*:

⟨Usr⟩ says ⟨Svc⟩ may use Email for Marketing?

The lower bound on behaviours is specified as phrases using the will verb. More specifically, the lower bound specified by the user preference is stated in terms of a will-*query*. Intuitively, a user asks the service to promise the obligatory behaviours. Here is an example of a will-query, in which the user requires the service to delete her email address within two years:

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

In the service policy, the lower bound is specified as a collection of will-*assertions*. The following assertion would satisfy the query above:

$$\langle \text{Svc} \rangle \text{ says } \langle \text{Svc} \rangle \text{ will delete Email within } 1\ \texttt{yr}.$$

Checking if a service policy satisfies a user preference is now 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. This is why the queries are evaluated against the union of the assertions in the policy *and* the preference.

2.1 SecPAL Overview

SecPAL4P is based on SecPAL, a language for writing authorization policies in decentralized systems [6, 11]. A concise and up-to-date formal specification of SecPAL can be found in [4]. This section provides a brief overview of the syntax of SecPAL.

A SecPAL authorization policy is a set of *assertions* α of the form

$$E \text{ says } f_0 \text{ if } f_1, ..., f_n \text{ where } c$$

where E is a constant[1], the f_i are *facts*, and c is a *constraint* on variables occurring in the assertion. A constraint is a formula from some domain-specific constraint language.

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 syntax for *facts* is defined below.[2] Henceforth, we keep to the following conventions for symbols: x, y denote variables, E, U, S constants from the set Const, e denotes an expression (i.e., either a variable or a constant), c a constraint, p a predicate, a an atom, f a fact, F a ground fact, α an assertion, $\mathcal{A}$ a set of assertions, θ a variable substitution, and γ a ground total variable substitution (mapping every variable to a constant). As usual, an atom is a

[1] Intuitively, E is of type principal, but for simplicity we omit types in this presentation, even though they would be used in a real implementation.

[2] For simplicity, we omit SecPAL's fact construct can say_0 here, as it is not needed in our examples.

predicate symbol applied to a tuple of expressions of the right arity. The predicate symbols are domain-specific, and we often write atoms in infix notation, e.g. `Alice` can read x, or y is a trusted seller.

$$\begin{array}{lrcl} \text{Fact} & f & ::= & a \\ & & | & e \text{ can say } f \end{array}$$

In an assertion $\alpha = \langle e \text{ says } f \text{ if } f_1, ..., f_n \text{ where } c \rangle$, e is the *issuer*, f the *head*, $f_1, ..., f_n$ the *body*, and c the *constraint* of α. The keyword if is omitted when $n = 0$; likewise, where c is omitted when $c = \text{true}$.

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

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

Let $\vec{x} = \langle x_1, ..., x_n \rangle$. We write $\exists \vec{x}(q)$ to denote $\exists x_1(...\exists x_n(q)...)$.

Queries are *evaluated* against sets of assertions. If $\mathcal{A}$ is a set of assertions and q a closed query, then we write $\mathcal{A} \vdash q$ iff q evaluates to true in the context of $\mathcal{A}$. We refer to [4, 6] for the formal definition of $\vdash$. A syntactic safety condition[3] ensures that query evaluation is sound and complete and terminates.

3 Example

Before we define the language and its semantics, we describe an extended example. In our example, the user Alice has an encounter with the service eBooking, which wants to collect her email address. Alice's privacy preference relating to

[3] Note that many of the assertions typically written in SecPAL4P would be unsafe according to the safety condition in [6] and [5]. They are, however, safe according to the more liberal IN/OUT-safety condition in [4].

her email address consists of five assertions and a query:

Pr.1 Alice says x may use Email for p if
x is a BookingSvc,
where $p \in \{\text{Confirmation}, \text{Newsletter}, \text{Stats}\}$

Pr.2 Alice says x may delete Email within t

Pr.3 Alice says x may send Email to y if
x is a BookingSvc,
y is a TrustedPartner

Pr.4 Alice says CA can say x is a y

Pr.5 Alice says x can say y is a TrustedPartner if
x is a BookingSvc

PrQ.6 Alice says $\langle$Svc$\rangle$ is a RegisteredSvc? $\wedge$
$\exists t$ ($\langle$Svc$\rangle$ says $\langle$Svc$\rangle$ will delete Email within t? $\wedge$ $t \leq$ 30 days?)

Pr.1 allows booking services to use Alice's email address for sending confirmations and newsletters, and for statistical purposes. Pr.2 permits any data collector to forget her email address. Pr.3 allows booking services to forward her email address to trusted partners. Pr.4 and Pr.5 use SecPAL's can say mechanism to express delegation of authority. Alice delegates authority over "is a"-facts to the certificate authority CA, and she allows booking services to define for themselves who their trusted partners are. PrQ.6 is a will-query. It specifies that Alice is only considering encounters with registered services, and only with services that promise they will forget about her address within a month.

The service eBooking has a policy consisting of three assertions and a query:

Pl.1 eBooking says eBooking will delete Email within 15 days

Pl.2 CA says eBooking is a RegisteredSvc

Pl.3 CA says eBooking is a BookingSvc

PlQ.4 $\langle$Usr$\rangle$ says eBooking may use Email for Confirmation? $\wedge$
$\langle$Usr$\rangle$ says eBooking may use Email for Stats? $\wedge$
$\langle$Usr$\rangle$ says eBooking may delete Email within 15 days?

In Pl.1, eBooking promises to delete email addresses within 15 days. Pl.2 and Pl.3 are credentials issued by CA, and PlQ.4 is a may-query in which the service declares that it may use email addresses for confirmation and statistical purposes, and that it may delete email addresses within 15 days.

To check if eBooking's policy satisfies Alice's preference, the placeholders $\langle$Usr$\rangle$ and $\langle$Svc$\rangle$ are replaced by Alice and eBooking, respectively, and the will-query and the may-query are evaluated against the union of all assertions. The first part of the will-query succeeds because of Pl.2 and Pr.4. The second part succeeds because of Pl.1. The may-query also succeeds because Pl.3 and Pr.4

together prove that eBooking is a booking service, and because of Alice's may-assertions Pr.1 and Pr.2. Thus the policy satisfies the preference, so by our correctness theorem 5.8 from Section 5, as long as eBooking indeed deletes the email address within the next 15 days and uses it for no other purposes but confirmation and statistics, eBooking's behaviour will also comply with Alice's preference.

Now suppose eBooking would like to share Alice's email address with eMarketing, one of its third-party partners. Since this is not within its policy, it will need to amend the policy and rerun the queries. First, it adds the following assertion:

eBooking says eMarketing is a TrustedPartner

Secondly, the amended may-query is the conjunction of the old may-query and the following may-query:

⟨Usr⟩ says eBooking may send Email to eMarketing?

Now, the unchanged will-query and the amended may-query have to be re-evaluated against the new collection of assertions, and indeed, the queries evaluate to true. However, this only proves that eBooking's action of sharing the email address with eMarketing is permitted. But we also must ensure that eMarketing's policy also satisfies Alice's preference. Suppose eMarketing's policy is as follows:

Pl′.1	eMarketing says eMarketing will delete Email within 30 days
Pl′.2	CA says eMarketing is a RegisteredSvc
PlQ′.3	⟨Usr⟩ says eMarketing may use Email for Marketing? ∧ ⟨Usr⟩ says eMarketing may delete Email within 30 days?

To check if eMarketing's policy satisfies Alice's preference, eBooking must have kept a copy of Alice's original, uninstantiated preference. This time, the placeholders ⟨Usr⟩ and ⟨Svc⟩ are instantiated by Alice and eMarketing, respectively, and the queries PrQ.6 and PlQ′.3 are evaluated against Alice and eMarketing's instantiated assertions. In this case, PlQ′.3 fails because Alice has not permitted her email address to be used for marketing purposes, so eBooking will refrain from sharing the email address with eMarketing.

4 SecPAL4P

This section defines SecPAL4P, an instance of SecPAL for specifying both users' preferences and services' policies.

In order to represent PII-relevant service behaviours, we fix a set of predicate symbols BehSymb. Atoms constructed from predicates in BehSymb are called *behaviour atoms*. These are often written in infix notation and may include atoms such as ⟨use FaxNo for Contact⟩ and ⟨delete Email within 1 yr⟩.

Henceforth, we write b to denote a behaviour atom, B for a ground behaviour atom, and $\mathcal{B}$ for a set of ground behaviour atoms.

We extend the syntax of facts by two constructs:

$$\begin{array}{lll} \text{Fact} \quad f & ::= & \ldots \\ & | & e \ \mathsf{may}\ b \\ & | & e \ \mathsf{will}\ b \end{array}$$

Definition 4.1. A *user-service pair* $\tau = (U, S)$ is a pair of constants denoting the *user* U and the *service* S during an encounter.

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

Definition 4.2. Let $\tau = (U, S)$ be a user-service pair.

- A τ-will-*query* is a query in which no subquery of the form S says S will b? occurs in the scope of a negation sign ($\neg$).
- A τ-may-*query* is a query in which no subquery of the form U says S may b? 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 bound or a lower bound on behaviours, such that the standard SecPAL query semantics matches this meaning. Disjunction and, similarly, existential quantification are allowed within a will-query, e.g.

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

In may-queries, however, disjunction does not make much sense. If a service wanted to state that it may possibly use the user's email address for contact or for marketing, it would specify a conjunctive query:

$$\begin{array}{l} U \ \mathsf{says}\ S \ \mathsf{may\ use}\ \text{Email}\ \mathsf{for}\ \text{Contact}? \ \wedge \\ U \ \mathsf{says}\ S \ \mathsf{may\ use}\ \text{Email}\ \mathsf{for}\ \text{Marketing}? \end{array}$$

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.

Definition 4.3. A τ-*preference* is a pair $(\mathcal{A}_{pr}, q_w)$ where $\mathcal{A}_{pr}$ is a set of assertions and q_w a closed τ-will-query. A τ-*policy* is a pair $(\mathcal{A}_{pl}, q_m)$ where $\mathcal{A}_{pl}$ is a set of assertions and q_m a closed τ-may-query.

As already mentioned informally, *satisfaction* between a policy and a preference is checked by evaluating the queries against the union of both the user's and the service's assertions:

Definition 4.4. A τ-policy $(\mathcal{A}_{pl}, q_m)$ *satisfies* a τ-preference $(\mathcal{A}_{pr}, q_w)$ iff $\mathcal{A}_{pl} \cup \mathcal{A}_{pr} \vdash q_m \wedge q_w$.

We assume that on an encounter between U and S, U provides a (U, S)-preference and 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 $\langle$Usr$\rangle$ and $\langle$Svc$\rangle$ that get instantiated with U and S, respectively. The formal sections in this document assume that such placeholders in preferences and policies have all been instantiated.

The definitions above formally specify an algorithm for checking if a policy satisfies a preference, but they do not show that the algorithm is correct. Indeed, there is as yet no definition of what we mean by *correct*. The following section formalises a notion of correctness and proves correctness of the satisfaction checking procedure.

5 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 a function **Beh** which maps each trace to a set of ground behaviour atoms. 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.)

Fig. 1 provides an overview of the formal notation introduced in this section.

Definition 5.1. 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 show how policies and preferences can be mapped to *sets* of traces. Furthermore, we will show that 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.

Then 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 5.2. 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.

5.1 Trace Semantics of Policies

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

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

Figure 1: Overview of the formal notation in Section 5

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

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

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

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

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

Definition 5.3. Let $\tau = (U, S)$, $\Pi_{pl} = (\mathcal{A}_{pl}, q_m)$ be a policy, and $\mathcal{A}$ a set of assertions. Then

$$\mathcal{B} \models^{\mathrm{pl}}_{\tau,\mathcal{A}} \Pi_{pl} \text{ iff } \mathcal{B} \models^{\mathrm{wa}}_{\tau,\mathcal{A}} \mathcal{A}_{pl} \text{ and } \mathcal{B} \models^{\mathrm{mq}}_{\tau,\mathcal{A}_{pl}\cup\mathcal{A}} q_m.$$

Let t be a trace. Then

$$t \models^{\mathrm{pl}}_{\tau,\mathcal{A}} \Pi_{pl} \text{ iff } \mathbf{Beh}(t) \models^{\mathrm{pl}}_{\tau,\mathcal{A}} \Pi_{pl}.$$

We write $[\![\Pi_{pl}]\!]^{\mathrm{pl}}_{\tau,\mathcal{A}}$ to denote the set of all traces t such that $t \models^{\mathrm{pl}}_{\tau,\mathcal{A}} \Pi_{pl}$.

5.2 Trace Semantics of Preferences

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

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

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 $\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_w$ holds if the behaviours in $\mathcal{B}$ include all behaviours specified as required by q_w, in the context of $\mathcal{A}$ (later, $\mathcal{A}$ will be instantiated to the assertions from both the service policy and the user preference). The relation is defined inductively as follows:

$\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} S \ \mathsf{says}\ S \ \mathsf{will}\ B?$	if	$\mathcal{B} \supseteq \{B\}$
$\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_1 \wedge q_2$	if	$\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_1$ and $\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_2$
$\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_1 \vee q_2$	if	$\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_1$ or $\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q_2$
$\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} \exists x(q)$	if	there exists $E \in \mathrm{Const}:\ \mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q[E/x]$
$\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}} q$	if	$\mathcal{A} \vdash q$ and no subquery of the form $S \ \mathsf{says}\ S \ \mathsf{will}\ B?$ occurs in q

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

Definition 5.4. Let $\tau = (U, S)$ be a user-service pair, $\Pi_{pr} = (\mathcal{A}_{pr}, q_w)$ a preference, and $\mathcal{A}$ a set of assertions. Then

$$\mathcal{B} \models^{\mathrm{pr}}_{\tau,\mathcal{A}} \Pi_{pr} \text{ iff } \mathcal{B} \models^{\mathrm{ma}}_{\tau,\mathcal{A}} \mathcal{A}_{pr} \text{ and } \mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}_{pr}\cup\mathcal{A}} q_w.$$

Let t be a trace. Then

$$t \models^{\mathrm{pr}}_{\tau,\mathcal{A}} \Pi_{pr} \text{ iff } \mathbf{Beh}(t) \models^{\mathrm{pr}}_{\tau,\mathcal{A}} \Pi_{pl}.$$

We write $[\![\Pi_{pr}]\!]^{\mathrm{pr}}_{\tau,\mathcal{A}}$ to denote the set of all traces t such that $t \models^{\mathrm{pr}}_{\tau,\mathcal{A}} \Pi_{pr}$.

5.3 Satisfaction and Compliance

Lemma 5.5 implies that checking a policy's may-query against a set of assertions is sufficient for guaranteeing that a set of behaviours respect the upper bound specified by the assertions, given that the set of behaviours also respect the upper bound specified by the may-query.

Lemma 5.5. 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 5.6 implies that checking a preference's will-query against a set of assertions is sufficient for guaranteeing that a set of behaviours respect the lower bound specified by the will-query, given that the set of behaviours also respect the lower bound specified by the assertions.

Lemma 5.6. 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 5.7 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 5.7. Let $\Pi_{pl} = (\mathcal{A}_{pl}, q_m)$ be a τ-policy and $\Pi_{pr} = (\mathcal{A}_{pr}, q_w)$ a τ-preference. If Π_{pl} satisfies Π_{pr} then $[\![\Pi_{pl}]\!]^{\text{pl}}_{\tau,\mathcal{A}_{pr}}$ is at least as strict as $[\![\Pi_{pr}]\!]^{\text{pr}}_{\tau,\mathcal{A}_{pl}}$.

Theorem 5.8 is our main correctness theorem. Given that a service trace complies with the service's own policy, successfully evaluating all queries is sufficient for guaranteeing that the trace also complies with the preference.

Theorem 5.8. Let t be a trace, $\Pi_{pl} = (\mathcal{A}_{pl}, q_m)$ a τ-policy and $\Pi_{pr} = (\mathcal{A}_{pr}, q_w)$ a τ-preference. If t complies with $[\![\Pi_{pl}]\!]^{\text{pl}}_{\tau,\mathcal{A}_{pr}}$ and Π_{pl} satisfies Π_{pr}, then t complies with $[\![\Pi_{pr}]\!]^{\text{pr}}_{\tau,\mathcal{A}_{pl}}$.

6 The PII-disclosing Protocol

We can now use the satisfaction checking algorithm and Theorem 5.8 as the main building block for the protocol for disclosing a PII.

User-service encounter The base case is an encounter between a user U and a service S (i.e., $\tau = (U, S)$), where S wishes to collect PII E from U. The protocol consists of the following steps:

1. U and S decide on a τ-preference Π_{pr} and a τ-policy Π_{pl}, respectively, to be used for this encounter.

2. If Π_{pl} does not satisfy Π_{pr}, then the protocol is aborted, and E is not disclosed. Otherwise, U discloses E to S. (Which principal performs the

satisfaction check has an impact on the trust model; possible choices would be U, S or a trusted third party. The most natural choice seems to be U, since the preference itself may be deemed at least somewhat confidential, whereas the service policy is usually public. Also, it is mainly in U's interest to control usage of the PII.)

3. S keeps a copy of Π_{pl} and Π_{pr} together with the data E. (This step is only needed when we allow S to change its policy after PII disclosure, or to disclose PIIs to third party services; see below.)

If E is disclosed, U can be assured that S will comply with her preference, assuming that S also complies with its own policy.

Policy change In practice, a service may wish to alter its policy regarding a PII even after it has already collected the PII. For example, a service may want to disclose the PII to a previously unknown third party at some point after the original encounter, even though the behaviour corresponding to the disclosure action was not declared in the may-assertions in its corresponding policy. Or it may wish not to exhibit a behaviour it had previously promised in the will-query of the policy.

Strictly speaking, both cases represent compliance violations of the service's own original policy. However, one could argue that such violations should be permitted as long as the new behaviours still comply with the user's preference. In this scheme, the service would need to alter its policy in such a way that the new behaviours comply with the new policy. It then has to check if the new policy still satisfies the preference. If it does not, then it must continue complying with the original policy; otherwise, it may continue complying with the new policy.

This scheme guarantees that all policy changes result in policies that still satisfy the user's preference.

Third party disclosure Once a PII has been collected by a service, it may or may not be further sent on to a third party service. In most scenarios, this action of disclosing a user's PII to a third party represents a relevant behaviour that should be controlled within preferences and policies. For example, the behaviour of forwarding a user's email address to eMarketing may be expressed by the behaviour atom $\langle$send Email to eMarketing$\rangle$.

However, controlling the action of disclosure is not sufficient. The intended property of such a system is that every service that receives a user's PII through a chain of disclosures also complies with the user's preference. To achieve this, a service S may only disclose a PII to a third party S' if

1. S's policy allows the disclosure, and

2. S' policy complies with U's preference. (Again, the trust model should dictate who performs this check. We believe that in most scenarios, it should S.)

The aforementioned *placeholders* ⟨Usr⟩ and ⟨Svc⟩ are important in this context. If PIIs may be forwarded along a chain of services, it is unreasonable to require that the original user preference contains specific references to all these services. Using the placeholders ⟨Usr⟩ and ⟨Svc⟩ effectively parameterizes preferences and policies by the current user-service pair τ. The placeholders are instantiated just before checking satisfaction. Also, service S must retain the original, uninstantiated preference along with the PII, so that it can later be instantiated using $\tau' = (U, S')$ when S prepares to disclose the PII to S'.

This scheme guarantees that all forwarding actions are permitted by U's preference and all recipients comply with U's preference (again assuming that they comply with their own respective policies).

7 Related work

This paper leverages SecPAL, a language originally designed for authorization, to specify privacy preferences and data-handling policies. SecPAL is particularly suitable because it supports extensible vocabularies and decentralized delegation of authority and is formally specified. It was also designed to achieve a good balance between expressiveness and usability. We refer to [4] for a more thorough and formal treatment of SecPAL, and to [5, 6] for background and general discussion on SecPAL as well as a review of other authorization languages such as ABLP [2], SPKI/SDSI [12], XrML [10], SD3 [16], RT [17], XACML [18], Cassandra [9], and DKAL [15].

The following discusses other languages designed for specifying privacy policies and preferences.

P3P and APPEL. P3P [20] specifies the privacy practices of web sites, i.e. how personal information is gathered and used. The user can specify his preferences in a separate language called APPEL [19], which is a syntactic pattern matcher for P3P policies. Alternatively, the user may do some ad-hoc processing of P3P policies, which may or may not comply with P3P's intended semantics. P3P and APPEL are strongly tied to the domain of web pages and web browsers, including cookies, the HTTP protocol, URLs, and thus have a fixed, built-in vocabulary of PII types, purposes, etc. P3P allows the specification of behaviour in case a site violates its declared policy.

P3P does not support general obligations such as ⟨will notify within 30 days⟩. The only obligations addressed by P3P are related to data retention. Behaviours in P3P cannot be conditioned on constraints or predicates. Checking if a P3P policy is satisfied by an APPEL preference is only informally described, and it is unclear what the matching algorithm guarantees: neither the P3P policy nor the APPEL preference have any formal semantics. Finally, P3P policies may define whether data can be forwarded to third parties but does not offer any control over third parties' data handling.

EPAL and EPAL-QUERY. EPAL [1] is a language for specifying a data handling policy inside an enterprise. The language EPAL-QUERY can be used to specify authorization queries to obtain data access. EPAL is a special-purpose language with only one predicate that allows describing common user data handling practices inside a large enterprise; it is not possible to extend the language with other predicates, purposes, etc. This predicate has a fixed number of arguments of fixed types like the handler category, the handling action, the data type, the purpose of handling and the obligation connected with the data. The predicate may have constraints expressed in terms of relations on numbers, strings, sets, and other data types. The query satisfaction is described as an informal algorithm.

Prime-DHP. The PRIME data handling policy [3] allows one single data handling predicate which specifies recipients (where each recipient is a set of conditions and a subject), handling actions, purposes and obligations (where a obligation is a set of triggers and actions). A policy is a template that is instantiated by the user. Forwarding users data from the first receiver to the second is handled, but data handling is not enforced by the second and more than two hops are not possible. Prime-DHP does not deal with the issue of user preferences, hence instantiation of the templates has to rely on unspecified ad-hoc mechanisms. Just like P3P and EPAL, Prime-DHP lacks a formal model, and only provides an XML-based syntax, but no syntax that can be both easily read by humans and processed by machines.

8 Discussion

SecPAL4P (or actually, the underlying language, SecPAL) 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 SecPAL4P using a dedicated editor with tool support, end users cannot be expected to be willing to learn the language. A number of different tools could be built on top of SecPAL4P to enhance usability for end users. Users could be offered to select amongst a small number of predefined 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 may let the user define exceptions to the predefined preferences. Users may also be able to download preferences provided by trusted third parties; also, 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.

The example preference and policies given in Section 3 were very small; real-world preferences and policies will usually contain many more assertions, for instance if there are many different types of PIIs for which preferences and policies have to be specified. One way to deal with this complexity is to use hierarchical predicate parameters [4], 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/Postcode, /Addr/Email/Secondary etc. for purpose p.

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 types) 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?
    [...]
}
```

This work was designed to be general-purpose; as such, we abstract away from a number of details that would need to be designed and implemented for a concrete application domain. First of all, one has to fix an application-specific vocabulary of SecPAL predicates; in particular, all relevant behaviours must be expressible in terms of some behaviour predicate.

Secondly, one needs to decide and agree on the semantics of these behaviours. In many cases, it may be sufficient to specify them informally, but even then it is important to pay attention to details. For example, consider the behaviour atom delete Email within 1 day. What does "delete" mean and entail? Who measures the time span, and when do you start counting? For a fully formal treatment, one needs to define the structure of traces and the behaviour mapping **Beh** from Section 5.

Thirdly, it has to be ensured that services actually adhere to their own policies. After all, the correctness theorem 5.8 rests on this assumption. This means that any behaviour it exhibits must be permitted by its may-query, and it must exhibit all behaviours promised by its will-assertions. Again, in most cases it will be unfeasible to enforce this by static analysis or dynamic monitoring. For small systems, formal methods and obligation-enforcing mechanisms may be applicable; we may investigate these in future work. A fully formal treatment would also require a mapping between real, concrete executions of a service to traces and behaviours.

Future work may also study options for the case when the policy does not satisfy the preference. When this occurs, the user may just decide not to use the service, but perhaps we could develop tools for negotiating an acceptable policies or for suggesting changes to the preference. Such tools may rely on previous work on abduction in the context of SecPAL [8, 7].

This work was presented in the context of privacy. However, usage control, i.e. attaching rights and obligations to data, is also useful in other scenarios such as enterprise right management. It would be interesting to study whether SecPAL4P is suitable for specifying licenses in such other contexts.

References

[1] Enterprise privacy authorization language, research report. Technical report, Nov. 2003. http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/index.html.

[2] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[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] M. Y. Becker. SecPAL formalisation and extensions. Technical Report MSR-TR-2009-127, Microsoft Research, 2009.

[5] M. Y. Becker, C. Fournet, and A. D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. Technical Report MSR-TR-2006-120, Microsoft Research, 2006.

[6] 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.

[7] M. Y. Becker, J. F. Mackay, and B. Dillaway. Abductive authorization credential gathering. In *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY'09)*, pages 1–8, 2009.

[8] 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.

[9] M. Y. Becker and P. Sewell. Cassandra: Flexible trust management, applied to electronic health records. In *IEEE Computer Security Foundations Workshop*, pages 139–154, 2004.

[10] ContentGuard. *eXtensible rights Markup Language (XrML) 2.0 specification part II: core schema*, 2001. At `www.xrml.org`.

[11] B. Dillaway and J. Hogg. *Security Policy Assertion Language (SecPAL) Specification, version 1.0*, February 2007.

[12] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory, RFC 2693, September 1999. At `www.ietf.org/rfc/rfc2693.txt`.

[13] European Parliament and the Council. Directive 95/46/EC - the data protection directive, October 1995.

[14] European Parliament and the Council. Directive 2002/58/EC on privacy and electronic communications, July 2002.

[15] Y. Gurevich and I. Neeman. DKAL: Distributed-knowledge authorization language. In *IEEE Computer Security Foundations Symposium*, pages 149–162, 2008.

[16] T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 106–115, 2001.

[17] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

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

[19] W3C. *A P3P Preference Exchange Language 1.0, Working Draft*, Apr. 2002. http://www.w3.org/TR/P3P-preferences.

[20] W3C. *P3P 1.1 Specification, Working Group Note*, Nov. 2006. http://www.w3.org/TR/P3P11.

A Proofs

Restatement of Lemma 5.5. 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 says S 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 says S may B? occurs in q_m. From (b), $\mathcal{B} = \emptyset$, hence trivially $\mathcal{B} \subseteq \mathcal{B}'$. □

Restatement of Lemma 5.6. 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 says S 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 5.7. Let $\Pi_{pl} = (\mathcal{A}_{pl}, q_m)$ be a τ-policy and $\Pi_{pr} = (\mathcal{A}_{pr}, q_w)$ a τ-preference. If Π_{pl} satisfies Π_{pr} then $[\![\Pi_{pl}]\!]^{\mathrm{pl}}_{\tau,\mathcal{A}_{pr}}$ is at least as strict as $[\![\Pi_{pr}]\!]^{\mathrm{pr}}_{\tau,\mathcal{A}_{pl}}$.

Proof. Suppose $t \in [\![\Pi_{pl}]\!]^{\mathrm{pl}}_{\tau,\mathcal{A}_{pr}}$. Let $\mathcal{B} = \mathbf{Beh}(t)$. We need to show that $t \in [\![\Pi_{pr}]\!]^{\mathrm{pr}}_{\tau,\mathcal{A}_{pl}}$, 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 5.5, $\mathcal{B} \models^{\mathrm{ma}}_{\tau,\mathcal{A}_{pl}\cup\mathcal{A}_{pr}} \mathcal{A}_{pl} \cup \mathcal{A}_{pr}$. 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 5.6 can be applied to get (b) $\mathcal{B} \models^{\mathrm{wq}}_{\tau,\mathcal{A}_{pl}\cup\mathcal{A}_{pr}} q_w$. □

Restatement of Theorem 5.8. Let t be a trace, $\Pi_{pl} = (\mathcal{A}_{pl}, q_m)$ a τ-policy and $\Pi_{pr} = (\mathcal{A}_{pr}, q_w)$ a τ-preference. If t complies with $[\![\Pi_{pl}]\!]^{\mathrm{pl}}_{\tau,\mathcal{A}_{pr}}$ and Π_{pl} satisfies Π_{pr}, then t complies with $[\![\Pi_{pr}]\!]^{\mathrm{pr}}_{\tau,\mathcal{A}_{pl}}$.

Proof. This is a corollary of Lemma 5.7 and Lemma 5.2. □