PSpec: A Formal Privacy Language for Data Analytics

Abstract

Business organizations regularly collect customer data to improve their services. Organizations may want to share data within themselves or even with third-parties to maximize data utility. Since business data contain lots of customer information, organizations must respect customers' privacy expounded by privacy laws. In this paper, we present the formal privacy specification language PSpec (Privacy Specification) for data usage restrictions. Comparing with previous works, PSpec specializes in data analytics, and also provides explicit support for data desensitization and association to balance data privacy and utility. We moreover present redundancy and conflict analysis algorithms to help data owners write PSpec policies. To evaluate PSpec and our policy analysis algorithms, we carry out a case study on TPC-DS benchmark and several experiments on synthetic rules. The results demonstrate the applicability of PSpec and the efficiency of the policy analysis algorithms.

1. Introduction

As more and more personal data are being collected by organizations, privacy issues are becoming increasingly important in modern society. In the past decade, privacy specification languages have been developed to formalize the text-based privacy policies. Since organizations may have different privacy requirements, it is important to have a flexible language to specify various privacy requirement across departments. Typical works include EPAL [3] and XACML [19]. Meanwhile, customer information collected by business organizations is invaluable. Organizations can make profits by sharing such information with third parties. For instance, manufacturers may predict sale trends by analyzing sales information of a small retail company at cost. Advertisers may also pay for user information in social networks. Opportunities are abundant if data can be shared with third parties. Demands for sharing business data with third parties are becoming more viable.

However, business data contain lots of private information such as identities and personal preferences. Misusing such data can cause severe privacy breaches and even violate privacy laws. More importantly, it immediately tarnishes the organization's reputation by attracting unwanted publicity. Business organizations could not be more concerned about protecting customers' privacy when they share custom information within themselves or with business partners. Privacy protection however is not easy [4, 22]. Several mechanisms have been proposed to protect privacy from adversaries.

A natural idea for privacy protection is by anonymization [28]. Through anonymization, data related to private information are removed or generalized. Privacy hence can be protected. Anonymization techniques however depend on data contents. Dates of birth and shopping histories, for instances, require different techniques. When the privacy and utility requirements change, anonymization techniques need be fine-tuned to fulfill new requirements. Anonymization subsequently can be tedious and cost-ineffective. Releasing anonymized data may be impractical for diverse business data and privacy requirements.

A more recent and promising privacy protection mechanism is differential privacy [9, 10]. In differential privacy, business data are not shared. A carefully designed curator resolves a pre-determined set of queries from third parties instead. Through noisy answers, one can argue that private information cannot be leaked through the curator. Lots of researches are devoted to designing curators for various data and queries. Designing differentially private curators however requires tedious analysis and mathematical insights. Adopting differential privacy in organizations with varying business data can be too challenging at present.

From data owners' viewpoint, a more desirable and practical approach is sharing information under privacy specifications. Similar to differential privacy, a curator is used to mediate data and queries. It however only enforces privacy specifications on queries. Privacy specifications instead protect privacy by depicting data usage restrictions. If a query does not access the data according to the restrictions in privacy specifications, the curator immediately rejects the query. Privacy specifications are written by legal experts to adhere to privacy laws and policies. The practical approach effectively shifts the responsibility of privacy protection to privacy specifications [27].

Privacy specifications may involve complicated conditions and operations. Consider the sales information of an online retail company. The company may be willing to share the amounts of all transactions last year. It may be hesitant to share the amounts of all transactions associated with an IP address last year. Nevertheless, the company may release such sensitive information after *desensitization*. For instance, transaction amounts associated with *truncated* IP addresses can be shared, as well as differentially private *average* transaction amounts associated with full IP addresses. Privacy specifications can be very complex and even confusing. A formal foundation is certainly needed.

In order to describe intricate privacy specification, we present the formal privacy language PSpec (Privacy Specification) for specifying data usage restrictions in data analytics. Its formal semantics help developers implement curators easily and correctly. PSpec moreover abstracts away details of underling data models and desensitize operations. Its specifications can be enforced by existing data analytics systems like Hive and Spark-SQL. PSpec moreover can employ techniques from differential privacy as desensitize operations to share aggregated information.

After privacy specifications are written, it is equally important to ensure that the specifications are semantically sound. Thanks to its formal semantics, PSpec specifications can be analyzed to identify potential flaws in privacy policies. Two useful analyses are redundancy analysis and consistency analysis [1]. Intuitively, a rule is *redundant* if it has no effect on the entire specification; a set of rules are *inconsistent* if they may issue restrictions that cannot be satisfied simultaneously. A redundant rule wastes computation resources. Inconsistent rules indicate contradictory privacy policies. Both should be identified and carefully reexamined. Yet privacy policies are interdependent on data association. Manually identifying such problems among intricate real-world privacy policies is hard. We develop novel techniques to analyze specifications using SMT solvers.

Inconsistency among privacy specifications allowing data association is not straightforward to formulate. Traditional inconsistency considers privacy rules with only one data category. When data analysts wish to access two different data, privacy rules for the two data categories are triggered. If requirements of the triggered rules cannot be satisfied simultaneously, this is inconsistency. For data analytics, privacy rules with multiple data categories are common. Different desensitize requirements can be specified for different combinations of data categories. When multiple data are accessed, several privacy rules with different desensitize requirements can be triggered. It is easy to have unsatisfiable desensitize requirements when data association is allowed. Such unsatisfiable requirements precisely indicate that the requested data cannot be desensitized to protect privacy. They should not be seen as (classical) inconsistency. For analysis with data association, inconsistency need be defined properly to avoid such false alarms. We propose a new definition for data analytics.

The main contributions of this paper are as follows:

- We present PSpec, a formal privacy language for data analytics with explicit support for data association and desensitization.
- We generalize the formulation of inconsistency for privacy policy specifications allowing data association.
- We present redundancy and inconsistency analysis algorithms for PSpec to analyze the PSpec policy.
- Finally, we carried out a case study on TPC-DS [21] benchmark and performed extensive experiments on synthetic rules.

The rest of the paper is organized as follows. Section 2 and Section 3 present the language design and formal semantics of PSpec respectively. Section 4 gives redundancy and conflict analyses for PSpec. Section 5 reports our primary implementation and evaluation. Section 6 further discusses related works. Finally, Section 7 concludes this paper.

2. PSpec Language

We consider the scenario with the following participants:

- The data *owner* such as a retail company, who shares data with third-party analysts under agreements.
- The data *analyst*, who performs data analysis over the shared data.
- The data analytics *system*, which manages the data and allows the analyst to submit queries for data analysis.

Queries submitted by the data analyst may violate the data owner's privacy policy. The primary goal of PSpec is to allow the legal team of the data owner to specify enforceable data usage restrictions such that only the privacy-compliant queries are authorized to access the data.

In the remainder of this section, we introduce the design of PSpec. As in EPAL [3], we separate PSpec into *vocabulary* and *policy*. Vocabulary defines a set of abstract concepts. Policy contains a set of PSpec rules.

2.1 Vocabulary

The data owner defines a set of user categories and data categories in vocabulary. A user category represents a role. A data category represents a privacy-related data concept. Both user and data categories are hierarchical. In a category hierarchy, a parent category is the generalization of its child categories; a child category is a specialization of its parent category. When a parent category is referred, all its descendants are also referred except those excluded explicitly.

Fig. 1 shows category hierarchies for a retail company as an example. The top-level user category is *Analyst* representing all data analysts. *Analyst* is divided into *Report Analyst*, *Marketing Analyst*, and *Advertise Analyst* based on the tasks assigned to them. Data categories are classified as *Key Attribute*, *Quasi Identifier*, and *Sensitive Attribute*. *Key Attribute* are the attributes that can uniquely identify an individual, such as *Name* and *Phone*. *Quasi Identifier* is the attributes that may identify individuals by association, such as the combination of *Birth*, *Zip*, and *Gender* [28]. Finally, *Sensitive Attribute* carries sensitive personal information, such as customers' sales records.

The data owner also specifies supported desensitize operations for data categories. Desensitize operations can be used to make sensitive data category ready for access. For example, desensitize operations for *Price* can be aggregate operations such as *Min*, *Max*, and *Avg*; they can be the *truncated* operation for *Zip*. A data category automatically inherits its ancestors' desensitize operations.

2.2 Policy

Before introducing details, we discuss data access issues pertaining to data analytics. In data analytics, accessing multiple pieces of information together may not be equivalent to accessing each piece separately because of *association*. For example, accessing a customer's address and salary separately may be acceptable. Yet accessing them together causes a severe privacy breach. Instead of forbidding accessing any sensitive information, *desensitization* is useful to balance between privacy and utility in data analytics. For instance, full IP addresses may locate individuals; *truncated* IP addresses may be acceptable and useful to data analysis.

The data owner defines a set of PSpec rules to regulate data access. Informally, each rule states under what restrictions can a user category access certain data categories together. We require a user category to satisfy all the applicable rules. The reason is that usually only a part of the business data is related to customers' privacy. This allows the data owner to focus on privacy-related data.

The rule grammar is shown in Fig. 2. A rule contains a *scope* and *restrictions* (separated by =>). The scope is defined by *User-Ref* and *Data-Assoc* (short for *Data Association*). *User-Ref* refers a user category (<user>) to specify applicable user categories. *Data-Assoc* specifies applicable association of data categories by a sequence of *Data-Ref*s. Each *Data-Ref* refers a data category (<data>) with an *action*. An *action* specifies how a data category is accessed. We distinguish two types of actions since they have different privacy implications. The action *projection* means that data categories are output by a query. The action *condition* denotes that data categories are used in the control flow of a query. The actions also form a hierarchy where *access* is the parent of *projection* and *condition*. For *Data-Assoc*, data categories referred in each *Data-Ref* must be disjoint. That is, no referred data category is the parent of another.

A rule can *forbid* the query or specify desensitize requirements by restrictions. A restriction is a sequence of *Desensitize* with the same length as the data association of the rule. Each *Desensitize* specifies a set of admissible desensitize operations (<operation>) required for the corresponding data category ($\emptyset$ denotes no desensitization requirement). The corresponding data category must support the specified desensitize operations as defined in vocabulary. A query must satisfy one of the restrictions in an applicable rule.

Example 2.1. Consider the privacy policy of a retail company. First, the company forbids any access to customer names by *Analyst*. We use the following PSpec rule:

r_1: `Analyst,[access Name]=>forbid`

Second, the company further requires the sales data should be aggregated when outputting with personal information.

r_2: `Analyst,[projection All exclude Sensitive Attribute,`
`projection Price]=>[{},{avg,max,min,sum}]`

In r_2, we define a data association of length two. The first data access refers to the projection on all but sensitive data categories. The second data access refers to the projection on sales price. The rule r_2 allows direct access to all but sensitive data but requires *Price* to be aggregated by one of the *avg*, *max*, *min*, or *sum* desensitize operations.

Finally, the company forbids any access to quasi-identifiers such as birth, gender, and zip together [28].

r_3: `Analyst,[access Birth,access Gender,access Zip]=>`
`forbid`

Note that r_3 only forbids access to them together. One can access any one or two of them freely.

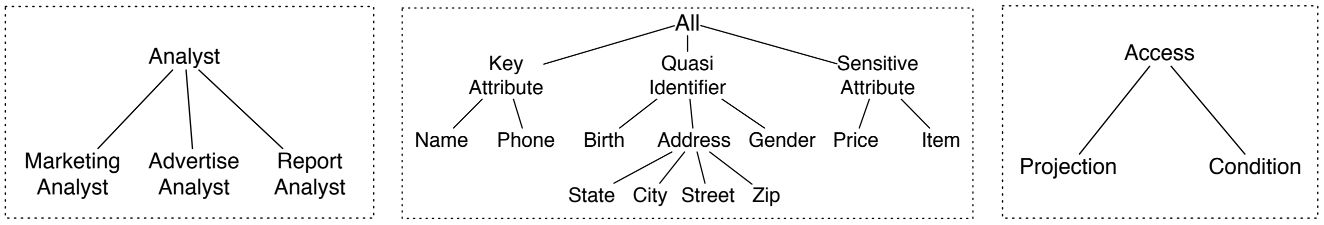
2.3 Discussion

A data category is desensitized when one of the specified desensitize operations is performed. We do not support sequences of desensitize operations since it would complicate PSpec rules and queries. To enforce sequences of desensitize operations, the data owner can define a new operation that combines all required operations. Also note that PSpec does not model *access purpose* directly since it would be difficult to verify the claimed purpose mechanically. We thus assume each data analyst is assigned a user category by her tasks.

PSpec is a privacy language for specifying data usage restrictions in data analytics. It is designed to be

- *usable* for legal teams with little technical background;
- *suitable* for specifying restrictions in data analytics; and
- *enforceable* by existing data analytics systems.

To achieve usability, PSpec is designed as a high level language based on abstract concepts to leave out details of the underling data models and queries. To be suitable for data analytics, PSpec provides explicit support for data association and desensitization. To attain enforceability, we only



(a) Example User Hierarchy (b) Example Data Hierarchy (c) Example Action Hierarchy

Figure 1: Example Category Hierarchies

```
Rule        ::= User-Ref, Data-Assoc =>
            ('forbid' | Restriction_1,..., Restriction_m)
User-Ref    ::= <user> ('exclude' <user>+)?
Data-Assoc ::= '[' Data-Ref_1,···,Data-Ref_n ']'
Data-Ref    ::= Action <data> ('exclude' <data>+)?
Action      ::= 'access' | 'projection' | 'condition'
Restriction::= '[' Desensitize_1,···,Desensitize_n ']'
Desensitize::= '{'(<operation>_1,···,<operation>_k)?'}'
```

Figure 2: PSpec Rule Grammar

include minimal elements in PSpec, and leave out others in privacy policies like *purpose*, *obligation*, and *condition*. However, it is straightforward to extend PSpec with these elements if necessary. To enforce the PSpec policy within a data analytics system, one simply needs to analyze each submitted query to identify how each data category is accessed with what operation, and check the compliance according to the semantics discussed in the next section.

3. Formal Semantics

In this section, we present the formal semantics of PSpec, which lay the foundation for policy enforcement and analysis. Recall that PSpec specifies restrictions on data analytics, thus the semantics are defined against queries.

3.1 Rule

Recall that on the data owner's side, a hierarchy is used to organize each of the user, data, and action categories. A policy rule may refer to internal nodes in hierarchies. In the following, we use the concepts associated with leaf categories of these hierarchies to define the semantics of a rule.

Let $\mathbb{U}_{user}$, $\mathbb{U}_{data}$ and $\mathbb{U}_{act}$ be the universal set of all user categories, data categories and action categories referred by leaf categories of the user hierarchy, the data hierarchy and the action hierarchy, respectively. We use $u \in \mathbb{U}_{user}$, $d \in \mathbb{U}_{data}$, and $a \in \mathbb{U}_{act}$ to denote a *leaf* concept of the user, data, and action categories, respectively. Especially, we denote the pair $\langle d, a\rangle$ a *data access*. Let $\mathbb{U}_{da} = \mathbb{U}_{data} \times \mathbb{U}_{act}$ be the set of all data accesses with respect to $\mathbb{U}_{data}$ and $\mathbb{U}_{act}$. An associated data access $\overline{\tau}$ of size n is an n-tuple $\overline{\tau} = (\tau_1, \tau_2, \dots, \tau_n)$, where $\tau_i \in \mathbb{U}_{da}$ for $1 \le i \le n$. The associated data accesses referred by a rule r must have the same size, denoted as $|r|$. Denote $\mathbb{U}_{da}^{|r|}$ the set of all associated data accesses of length $|r|$.

Given a rule r, we use $r.User \subseteq \mathbb{U}_{user}$ and $r.Asso \subseteq \mathbb{U}_{da}^{|r|}$ to denote the set of user categories and the set of associated data accesses applicable to r, respectively. In the grammar of Fig. 2, $r.User$ is specified by *User-Ref*. It corresponds to the leaves of the subtree rooted in the specified data category minus the leaves of the subtree rooted in any of the excluded data categories. The set $r.Asso$ is specified by *Data-Assoc* in the grammar. Let $r.Asso_i$ be the set of data accesses specified by the i-th *Data-Ref* of *Data-Assoc*, $r.Asso$ is the Cartisian product of $r.Asso_i$ for $1 \le i \le n$.

Let $\mathbb{U}_{op}$ be the universal set of all available desensitize operations. We use $\Delta \subseteq \mathbb{U}_{op}$ to denote a set of desensitize operations, and $\delta \in \mathbb{U}_{op}$ a desensitize operation. The *empty* desensitize operation is denoted by $\delta_\emptyset \in \mathbb{U}_{op}$. Given an associated data access $\overline{\tau} = (\langle d_1, a_1\rangle, \langle d_2, a_2\rangle, \dots, \langle d_n, a_n\rangle)$ of size n, a restriction $\overline{\xi}$ is an n-tuple $(\Delta_1, \Delta_2, \dots, \Delta_n)$, where $\Delta_i \subseteq \mathbb{U}_{op}$ specifies the set of desensitize operations. An operation in Δ_i must be taken to protect d_i against a_i. Without loss of generality, we assume the rule r contains m restrictions.

Given a rule r, we use $r.Res$ to denote the set of restrictions specified in r. Each restriction $\overline{\xi}_i \in r.Res$ is specified by *Restriction_i* in the grammar.

Definition 3.1 (Rule)**.** A rule r is a triple $r = (r.User, r.Asso, r.Res)$, where

- $r.User \subseteq \mathbb{U}_{user}$ is the set of user categories,
- $r.Asso \subseteq \mathbb{U}_{da}^{|r|}$ is the set of associated data accesses, and
- $r.Res = \{\overline{\xi}_1, \overline{\xi}_2, \dots, \overline{\xi}_m\}$ is the set of restrictions. Especially, $r.Res = \emptyset$ if r is a forbidden rule.

$r.User$ and $r.Asso$ together defines the *scope* of the rule r, written $scope(r) = (r.User, r.Asso)$. Informally, if a user in $r.User$ sends a query to perform an associated data access in $r.Asso$, the restriction $r.Res$ applies.

Example 3.1. Consider the following rule:

```
r_4: Report Analyst,[projection Address,projection Price]
                              =>[{},{sum,avg,min,max}]
```

Obviously, $|r_4| = 2$, and

$$\begin{aligned} r_4.User &= \{\text{Report Analyst}\} \\ r_4.Asso &= \{(\langle \text{State, projection}\rangle, \langle \text{Price,projection}\rangle), \\ &\quad (\langle \text{City, projection}\rangle, \langle \text{Price,projection}\rangle), \\ &\quad (\langle \text{Street, projection}\rangle, \langle \text{Price,projection}\rangle), \\ &\quad (\langle \text{Zip, projection}\rangle, \langle \text{Price,projection}\rangle)\} \\ r_4.Res &= \{(\mathbb{U}_{op}, \{\text{sum,avg,min,max}\})\} \end{aligned}$$

Note that $\mathbb{U}_{op}$ is used if there is no desensitize requirement.

3.2 Query

Similar as for a rule, we use the *leaf* categories to define the semantics of a data analytics query (or a query for short). This is not a limitation in practice. Non-leaf categories can be easily decomposed into a set of leaf categories.

The semantics of a query is defined from the information flow point of view [25, 26]. A query q is divided into a set of data leakages. A *data leakage* is a pair $((d, a), \delta)$ where (d, a) is a data access and δ a desensitize operation. A data leakage $((d, a), \delta)$ means that the data d is leaked through the action channel a after the desensitize operation δ. Let $|q|$ be the number of data leakages in q. We extract the data accesses and desensitize operations from the data leakages of q, each as a $|q|$-tuple.

Definition 3.2 (Query)**.** A query q is a triple $q = (q.user, q.asso, q.des)$, where

- $q.user \in \mathbb{U}_{user}$ is the user who submits q,
- $q.asso \in \mathbb{U}_{da}^{|q|}$ is the associated data accesses in q, and
- $q.des = (\delta_1, \delta_2, \ldots, \delta_{|q|})$ is the desensitize operations.

Example 3.2. Consider a report analyst submits a query:

```
q: SELECT State, City, avg(Price) from t_addr
```

where "t_addr" is a table name. Obviously, $|q| = 3$, and

$$\begin{aligned} q.user &= \text{Report Analyst}, \\ q.asso &= (\langle \text{State, projection}\rangle, \langle \text{City, projection}\rangle, \\ &\quad \langle \text{Price, projection}\rangle), \\ q.des &= (\delta_\emptyset, \delta_\emptyset, \text{avg}). \end{aligned}$$

Note that $\delta_\emptyset$ is used if one wants to access data without any desensitization.

Definition 3.3. Given two associated data accesses $\overline{\tau}$ and $\overline{\tau}'$, we say $\overline{\tau}$ is *included* in $\overline{\tau}'$ (written $\overline{\tau} \preceq \overline{\tau}'$) if all data accesses contained in $\overline{\tau}$ are also contained in $\overline{\tau}'$.

If we treat both $\overline{\tau}$ and $\overline{\tau}'$ as a set, $\overline{\tau} \preceq \overline{\tau}'$ just means $\overline{\tau}$ is a subset of $\overline{\tau}'$. Note that a smaller-size associated data access usually means larger scope.

Definition 3.4. A rule r is *applicable* to a query q (or equivalently, q *triggers* r) if

- $q.user \in r.User$, and
- there exists at least one associated data access $\overline{\tau}$ in $r.Asso$, such that $\overline{\tau} \preceq q.asso$.

The first condition in above definition performs user scope check, and the second condition performs the scope check of associated data access. The rule is applicable to the query only if both checks are passed. For the ease of following discussion, we say a user category u triggers a rule r if $u \in r.User$, and an associated data access $\overline{\tau}$ triggers r if there exists some $\overline{\tau}_r \in r.Asso$ such that $\overline{\tau}_r \preceq \overline{\tau}$.

Moreover, if there exists an $\overline{\tau}$ in $r.Asso$ such that the second condition holds, we further define $q.des|_{\overline{\tau}}$ as the projection of $q.des$ to $\overline{\tau}$, obtained by removing the desensitize operations in $q.des$ whose corresponding data access does not occur in $\overline{\tau}$. The size of the tuple $q.des|_{\overline{\tau}}$ equal to $|r|$. Note that there may be more than one $\overline{\tau}$ satisfying the second condition holds. With different $\overline{\tau}$, the projection $q.des|_{\overline{\tau}}$ may be different. We thus define

$$q.des|_r \;=\; \{q.des|_{\overline{\tau}} \,|\, \overline{\tau} \;\in\; r.Asso \text{ and } \overline{\tau} \;\preceq\; q.asso\}$$

Definition 3.5. A n-tuple of desensitize operations $\overline{\delta} = (\delta_1, \delta_2, \ldots, \delta_n)$ *satisfies* a restriction $\overline{\xi} = (\Delta_1, \Delta_2, \ldots, \Delta_n)$ if $\delta_i \in \Delta_i$ for $1 \leq i \leq n$.

Definition 3.6 (Rule Satisfaction)**.** A query q *satisfies* a rule r iff either r is not applicable to q or for any $\overline{\delta} \in q.des|_r$, there exists at least one restriction $\overline{\xi} \in r.Res$, such that $\overline{\delta}$ *satisfies* $\overline{\xi}$.

Intuitively, a query satisfies a triggered rule if the data categories accessed by the query are properly desensitized.

Example 3.3. Consider the rule r_4 in Example 3.1 and the query q in Example 3.2. r_4 is applicable to q since $q.user \in r_4.User$ and two former associated data accesses in $r_4.Asso$ are included in $q.asso$. Hence $q.des|_{r_4} = \{(\delta_\emptyset, \text{avg})\}$. Since $(\delta_\emptyset, \text{avg})$ satisfies the constraint in $r_4.Res$, r_4 is satisfied by q (Definition 3.6).

Definition 3.7 (Policy Satisfaction)**.** A query q *satisfies* a PSpec policy iff q *satisfies* all rules of the policy.

3.3 Symbolic Encoding

Since our policy analysis algorithms heavily rely on SMT solving techniques, we present in this subsection an symbolic encoding scheme for PSpec rules.

We use a tuple of $|r|$ variables $(x_r^1, x_r^2, \ldots, x_r^{|r|})$ to denote the associated data accesses of a rule r, where $x_r^i \in \mathbb{U}_{op}$ is called a *rule variable* for $1 \leq i \leq |r|$.

Definition 3.8. Let $X_r = \{x_r^1, x_r^2, \ldots, x_r^{|r|}\}$ be the set of *rule variables* of r. A restriction $\overline{\xi} = (\Delta_1, \Delta_2, \ldots, \Delta_{|r|})$ in r can be symbolically encoded as a logical formula over X_r

$$\phi_{\overline{\xi}} = \bigwedge_{1 \leq i \leq |r|} (x_r^i \in \Delta_i)$$

The *restriction formula* of a rule r is

$$\phi_r = \bigvee_{\forall \overline{\xi} \in r.Res} \phi_{\overline{\xi}}$$

Especially, $\phi_r = false$, if r is a forbidden rule.

Note that a tuple $\overline{\delta}$ of desensitize operations of length $|r|$ can be considered as a valuation to the variables X_r. Then we have the following lemma.

Lemma 1. A tuple of desensitize operations $\overline{\delta}$ *satisfies* one of the restrictions in a rule r iff $\phi_r(\overline{\delta})$ evaluates to true.

Example 3.4. Consider the rule r_4 in Example 3.1, where $|r_4| = 2$. Let x_r^1, x_r^2 be rule variables of r_4. The logical formula of r_4 is

$$\begin{aligned}\phi_{r_4} &= (x_r^1 \in \mathbb{U}_{op}) \wedge (x_r^2 \in \{\text{sum, avg,min,max}\}) \\ &= x_r^2 \in \{\text{sum, avg,min,max}\}\end{aligned}$$

The desensitize operations executed by the query q in Example 3.2 is $(\delta_\emptyset, \text{avg})$. It does satisfy the constraint, since $\phi_{r_4}(\delta_\emptyset, \text{avg}) = true$.

4. Policy Analysis

After introducing the formal semantics, we discuss some useful policy analysis algorithms for PSpec, namely redundancy analysis and consistency analysis.

4.1 Redundancy Analysis

Since a query should satisfy all rules in a PSpec policy, some rules may thus be redundant. The redundant rules not only cost unnecessary time for policy enforcement, but also may represent potential errors or unintended side-effects in a policy [1]. In this subsection, we discuss redundancy analysis for the PSpec policy.

Definition 4.1 (Rule Redundancy)**.** A rule r' is said *redundant* with respect to another rule r if for any query q such that q satisfies r, q also satisfies r'.

With the inclusion relation between associated data accesses (Definition 3.3), we define a similar relation between the scopes of two rules. Note again that a less-size associated data access usually means larger scope.

Definition 4.2. Given two rules r and r', we say the scope of r *subsumes* the scope of r', denoted as $scope(r) \sqsupseteq scope(r')$, if $r'.User \subseteq r.User$, and

$$\forall \overline{\tau}' \in r'.Asso, \exists \overline{\tau} \in r.Asso, \text{ s.t. } \overline{\tau} \preceq \overline{\tau}'.$$

Lemma 2. Given two rules r and r' with $scope(r) \sqsupseteq scope(r')$, we have $|r| \leq |r'|$; and for any query q, if r' is triggered, r is also triggered.

Moreover, we can further facilitate the check of scope inclusion with the following lemma:

Lemma 3. Given two rules r and r' such that $r'.User \subseteq r.User$, $scope(r) \sqsupseteq scope(r')$ iff

$$\forall 1 \leq i \leq |r|, \exists 1 \leq j \leq |r'|, \text{ s.t. } r.Asso_i \supseteq r'.Asso_j.$$

Example 4.1. Consider r_4 in Example 3.1 and the following rule:

```
r5:Analyst,[Access Price]=>[{avg,sum}]
```

We have $|r_5| = 1$, and

$$\begin{aligned} r_5.User =& \{\text{Market\_Analyst, Advertise\_Analyst,} \\ & \text{Report\_Analyst}\} \\ r_5.Asso =& \{(\langle \text{Price, projection}\rangle), (\langle \text{Price, condition}\rangle)\}\end{aligned}$$

Comparing r_5 to r_4, we have $r_4.User \subseteq r_5.User$; and for all associated data accesses $\overline{\tau}'$ in $r_4.Asso$, there exists an associated data access $\overline{\tau} = (\langle \text{Price, projection}\rangle)$ in $r_5.Asso$, such that $\overline{\tau} \preceq \overline{\tau}'$. Thus $scope(r_5) \sqsupseteq scope(r_4)$ (Definition 4.2). Obviously, whenever *Report Analyst* outputs *Address* and *Price* together (r_4 is triggered), r_5 must be triggered as well.

Definition 4.2 makes it possible to compare the scopes of two rules. We now compare the restrictions of two rules. The basic idea is to check their restriction formulas to see if one restriction implies another one. The hardness is that these two formulas are defined over two sets of rule variables. We thus need a mechanism to link these variables together.

Definition 4.3. Let r and r' be two rules, X_r and $X_{r'}$ their sets of rule variables, ϕ_r and $\phi_{r'}$ their restriction formulas, respectively. For any variable $x_r^i \in X_r$ $(1 \leq i \leq |r|)$ and $x_{r'}^j \in X_{r'}$ $(1 \leq j \leq |r'|)$, we say x_r^i *matches* $x_{r'}^j$ iff for any $\overline{\tau}' \in r'.Asso$, there exists $\overline{\tau} \in r.Asso$, such that $\overline{\tau}'[j] = \overline{\tau}[i]$, i.e., $r.Asso_i \supseteq r.Asso_j$. We denote all matched variables in r' for x_r^i as $matched(x_r^i, r')$.

Definition 4.4. Given two rules r and r' that $scope(r) \sqsupseteq scope(r')$, a *variable mapping* from r to r' is a function $\theta_{r'}^r : X_r \to X_{r'}$ such that $\forall x_r^i \in X_r, \theta_{r'}^r(x_r^i) \in matched(x_r^i, r')$.

Note that the mapping from X_r to $X_{r'}$ is possible since $|r| \leq |r'|$. As a convention, we denote all possible variable mappings from r to r' as $\Theta_{r'}^r$. Let ϕ_r be the restriction formula of r, we denote $\theta_{r'}^r(\phi_r)$ the formula obtained by replacing each x_r^i in ϕ_r with $\theta_{r'}^r(x_r^i)$ for $1 \leq i \leq |r|$.

Definition 4.5 (Rule Implication)**.** Given two rules r and r', we say r *implies* r' (denoted as $r \implies r'$), if

- $scope(r) \sqsupseteq scope(r')$, and
- $\forall x_{r'}^1 \ldots x_{r'}^{|r'|}.(\bigwedge_{\theta_{r'}^r \in \Theta_{r'}^r} \theta_{r'}^r(\phi_r)) \to \phi_{r'}$.

The following theorems state some properties of rule implication.

Theorem 1. Implication relation over rules is a partial order.

Theorem 2. A rule r' is redundant with respect to another rule r iff $r \implies r'$.

Example 4.2. Consider the rules r_4 and r_5 in Example 4.1. From previous discussion, we already know $scope(r_5) \sqsupseteq scope(r_4)$. Let X_{r_5}=$\{x^1_{r_5}\}$ and $X_{r_4} = \{x^1_{r_4}, x^2_{r_4}\}$, the restriction formulas of r_5 and r_4 are

$$\phi_{r_5} = x^1_{r_5} \in \{avg, sum\}$$
$$\phi_{r_4} = x^2_{r_4} \in \{avg, sum, min, max\}$$

Observering $r_5.Asso$ and $r_4.Asso$, we know that $x^1_{r_5}$ matches only $x^2_{r_4}$. The only variable mapping is $\theta^{r_5}_{r_4} = \{x^1_{r_5} \rightarrow x^2_{r_4}\}$. Thus, to check whether $r_5 \implies r_4$, it suffices to check the following formula:

$$\forall x^1_{r_4}, x^2_{r_4}.\ x^2_{r_4} \in \{\text{avg, sum}\} \rightarrow x^2_{r_4} \in \{\text{avg, sum, min, max}\}$$

Obviously, the above formula is true, and thus we have $r_5 \implies r_4$. From Theorem 2, we know r_4 is redundant w.r.t. r_5.

With the definitions above, it is straightforward to derive a redundancy analysis algorithm for the PSpec policy. Specially, for each pair of rules r and r', if $r' \implies r$ then r is reported redundant with respect to r'. Due to its simplicity, the details are omitted here.

4.2 Consistency Analysis

After discussing redundancy analysis for PSpec, we now introduce consistency analysis. Since we require all rules in a policy be satisfied, it is thus possible that some rules can never be satisfied simultaneously, which leads to inconsistency.

4.2.1 Consistency Definition

Data association makes the consistency analysis much more subtle and complicated. Traditional consistency considers privacy rules with only one data category. With this traditional definition, two rules are inconsistent if they can be triggered by a same query, while their restrictions cannot be satisfied simultaneously. To directly extend the classical consistence definition to analysis with data associations, we have the following definition.

Definition 4.6 (Weak Inconsistency)**.** Two rules r and r' are *weak inconsistent* iff there exist a user category u and an associated data access $\overline{\tau}$ such that $\overline{\tau}$ is *inaccessible* for u w.r.t. r and r', i.e., there is no query q such that $q.user = u$, $q.asso = \overline{\tau}$ and q satisfies both r and r'.

This *weak inconsistency* definition is problematic for privacy language. Consider following two rules:

```
r6: Analyst,[project Zip,access Price] => [{},{min,max}]
r7: Analyst,[access Birth,access Price] => [{},{avg}]
```

Clearly, when *Analyst* accesses *Zip*, *Birth* and *Price* together, there is no way to satisfy r_6 and r_7 simultaneously. But in practice, this may just be the user's intention. Note that data associations with various length usually have different privacy implications. From the point view of privacy protection, if a query attempts to access more data categories, it turns to be more dangerous. Thus the data owner may specify above two rules on purpose to forbid the simultaneous access of these three data categories.

The *weak inconsistency* definition also has efficiency problem. To decide whether two rules are weak inconsistent, one needs to check not only the associated data accesses in the scope of these rules, but also others (for example, the associated access to *Address*, *Birth* and *Price* in above example). Given N data categories and M ($M = 2$ PSpec) action categories, the total number of different associated data access is as large as $2^N * M$. The efficiency problem is further worsened when we consider the inconsistency of a set of, rather than two, rules. Thus, from the implementation point of view, weak inconsistency can lead to a very inefficient analysis algorithm.

We propose a novel inconsistency definition for analysis with data association. The basic idea is to predetermine a seed rule, and then define the consistency with respect to this seed rule.

Definition 4.7. A rule r is *inconsistent* with respect to an non-forbidden rule r_s iff there exist a user category $u_s \in r_s.User$ and an associated data access $\overline{\tau}_s \in r_s.Asso$ such that $\overline{\tau}_s$ is inaccessible for u_s w.r.t. r_s and r. The rule r_s is called the *seed rule*.

The above definition can be naturally extended to sets.

Definition 4.8 (Inconsistency)**.** A rule set R is *inconsistent* w.r.t. r_s iff there exist a user category $u_s \in r_s.User$ and an associated data access $\overline{\tau}_s \in r_s.Asso$ such that $\overline{\tau}_s$ is inaccessible for u_s w.r.t. the rules in $R \cup \{r_s\}$.

Intuitively, for a non-forbidden rule r_s, the defined data accesses $\overline{\delta}_s \in r_s.Asso$ should be accessible if properly desensitized. However, accessing $\overline{\delta}_s$ could trigger other rules, which may in turn render $\overline{\delta}_s$ inaccessible and lead to inconsistency. In practice, we are only interested in the minimal inconsistent rule set as follows.

Definition 4.9 (Minimal Inconsistency)**.** A rule set R is minimally inconsistent w.r.t. r_s if:

1. R is inconsistent w.r.t. r_s, and
2. $\forall R' \subset R$, R' is consistent w.r.t. r_s.

Example 4.3. Consider the rules r_6 and r_7. No matter r_6 or r_7 being the seed rule, there is no associated data access in the seed rule that triggers another rule. Rules r_6 and r_7 are thus consistent.

4.2.2 Consistency Analysis as Satisfiability Problems

In this subsection, we formulate the consistency analysis as satisfiability problems.

Definition 4.10 (Candidate Rule)**.** Given a seed rule r_s, we say a rule r is a *candidate rule* of r_s if there exist a user category $u_s \in r_s.User$ and an associated data access $\overline{\delta}_s \in r_s.Asso$ such that both u_s and $\overline{\delta}_s$ trigger r. Denote $cand(r_s)$ the set of all candidate rules of r_s.

Apparently, for any minimal inconsistent rule set R w.r.t. a seed rule r_s, $R \subseteq cand(r_s)$.

Definition 4.11. Given a seed rule r_s and a non-forbidden rule set $R \subseteq cand(r_s)$, $trig^*_{\overline{\delta}}(r_s, R)$ is the set of associated data accesses in r_s that trigger all rules in R.

Let r_s be a seed rule, and r a rule such that $r \in cand(r_s)$. Let X_r and X_{r_s} be the set of rule variables of r and r_s, respectively. If an associated data access $\overline{\tau}_s$ of r_s triggers r, there must exist an associated data access $\overline{\tau}$ of r, such that $\overline{\tau} \preceq \overline{\tau}_s$ (Definition 3.4). Then we can construct an interval mapping $\mathcal{I}^{\overline{\tau}}_{\overline{\tau}_s}$ from $[1, |r|]$ to $[1, |r_s|]$ as: $\forall i \in [1, |r|], \mathcal{I}^{\overline{\tau}}_{\overline{\tau}_s}[i] = j \in [1, |r_s|]$ such that $\overline{\tau}[i] = \overline{\tau}_s[j]$.

Definition 4.12. Let $\overline{\tau} \in r.Asso$ and $\overline{\tau}_s \in r_s.Asso$ be two associated data accesses that satisfy $\overline{\tau} \preceq \overline{\tau}_s$, a *variable mapping* with respect to $\overline{\tau}$ and $\overline{\tau}_s$ is a function $\vartheta^{\overline{\tau}}_{\overline{\tau}_s} : X_r \to X_{r_s}$ such that $\forall x^i_r \in X_r, \vartheta^{\overline{\tau}}_{\overline{\tau}_s}(x^i_r) = x^j_{r_s}$ with $j = \mathcal{I}^{\overline{\tau}}_{\overline{\tau}_s}(i)$.

Let ϕ_r be the restriction formula of r, we denote $\vartheta^{\overline{\tau}}_{\overline{\tau}_s}(\phi_r)$ the formula obtained by replacing x^i_r with $\vartheta^{\overline{\tau}}_{\overline{\tau}_s}(x^i_r)$ for $1 \leq i \leq |r|$.

Definition 4.13. The *association formula* for an associated data access $\overline{\tau}_s \in trig^*_{\overline{\delta}}(r_s, R)$ is

$$\psi_{\overline{\tau}_s} = \phi_{r_s} \wedge \Big(\bigwedge_{\forall r \in R} \; \bigwedge_{\forall \overline{\tau} \in r.Asso}^{\overline{\tau} \preceq \overline{\tau}_s} \vartheta^{\overline{\tau}}_{\overline{\tau}_s}(\phi_r) \Big)$$

where ϕ_{r_s} and ϕ_r are restriction formulas of r_s and r, respectively, $\vartheta^{\overline{\tau}}_{\overline{\tau}_s}$ is the variable mapping with respect to $\overline{\tau}$ and $\overline{\tau}_s$.

If $\psi_{\overline{\tau}_s}$ is unsatisfiable, $\overline{\tau}_s$ is just an associated data access that is inaccessible w.r.t. the rules in $R \cup \{r_s\}$. In other words, if $\psi_{\overline{\tau}_s}$ is unsatisfiable, the rule set R is inconsistent with respect to r_s (Definition 4.8).

Theorem 3. Given a seed rule r_s and a non-forbidden rule set $R \subseteq cand(r_s)$, the rule set R is *minimal inconsistent* w.r.t. r_s iff the following holds:

- $r_s.User \cap (\bigcap_{\forall r \in R} r.User) \neq \emptyset$,
- there exists $\overline{\tau}_s \in trig^*_{\overline{\delta}}(r_s, R)$ such that its association formula $\psi_{\overline{\tau}_s}$ is unsatisfiable, and
- $\forall R' \subset R$, R' is consistent w.r.t. r_s.

Remark that $\overline{\tau}_s \in trig^*_{\overline{\delta}}(r_s, R)$ in the second condition is important. If $\overline{\tau}_s \notin trig^*_{\overline{\delta}}(r_s, R)$, there must exist a rule r in R that is not triggered by $\overline{\tau}_s$. Let $R' = R \setminus \{r\}$. Then from the unsatisfiability of $\psi_{\overline{\tau}_s}$, we can never conclude that R is minimal inconsistent (since R' is inconsistent and $R' \subset R$).

Corollary 1. Given a seed rule r_s and a non-forbidden rule set $R \subseteq cand(r_s)$, if either $r_s.User \cap (\bigcap_{\forall r \in R} r.User) = \emptyset$, or $trig^*_{\overline{\delta}}(r_s, R) = \emptyset$. then the rule set R and all its supersets cannot be minimal inconsistent w.r.t. r_s.

Example 4.4. Consider the rule r_6 and the following rule:

```
r8: Analyst,[projection Price]=>[{avg,sum}]
```

where r_6 is set as the seed rule. Obviously, $r_6.User \cap r_8.User \neq \emptyset$. Let u_s be any user category in $r_6.User \cap r_8.User$, and $\overline{\tau}_s = (\langle Zip, projection\rangle, \langle Price, projection\rangle)$ in $r_6.Asso$. The only associated data access $\overline{\tau} = (\langle Price, projection\rangle)$ in $r_8.Asso$ satisfies $\overline{\tau} \preceq \overline{\tau}_s$. By Definition 3.4, u_s and $\overline{\tau}_s$ together trigger r_8. Thus r_8 is a candidate rule of r_6.

Let $x^1_{r_6}, x^2_{r_6}$ be rule variables of r_6, and $x^1_{r_8}$ the rule variable of r_8. Apparently,

$$\phi_{r_6} = true \wedge (x^2_{r_6} \in \{min, max\})$$
$$\phi_{r_8} = x^1_{r_8} \in \{avg, sum\}.$$

Moreover, $trig^*_{\overline{\delta}}(r_6, \{r_8\}) = \{\overline{\tau}_s\}$. The variable mapping w.r.t. $\overline{\tau}$ and $\overline{\tau}_s$ is $\vartheta^{\overline{\tau}}_{\overline{\tau}_s} = \{x^1_{r_8} \to x^2_{r_6}\}$. The association formula for $\overline{\tau}_s$ is

$$\psi_{\overline{\tau}_s} = (true \wedge (x^2_{r_6} \in \{\text{min, max}\}) \wedge (x^2_{r_6} \in \{\text{avg, sum}\}).$$

Apparently $\psi_{\overline{\tau}_s}$ is unsatisfiable, thus the rule set $\{r_8\}$ is minimal inconsistent w.r.t. r_6.

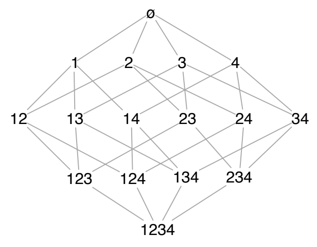
4.2.3 Analysis Algorithm

The purpose of inconsistency analysis on a policy is to find all minimal inconsistency rule sets w.r.t. each non-forbidden rule (as the seed rule) in the policy.

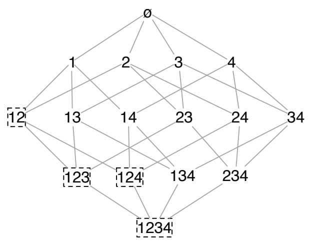
According to Definition 4.9, to find all minimal inconsistent rule sets w.r.t. a seed rule r_s, it suffices to only consider the subsets of the candidate rules $cand(r_s)$ of r_s. Moreover, all subsets of the candidate rules $cand(r_s)$ constitute a set containment lattice. An example lattice of four candidate rules is shown in Fig. 3a, where each node represents a subset of these candidate rules. We apply the *levelwise search* technique [18] to find in this lattice all minimal inconsistent rule sets. The efficiency of the levelwise search process relies on the search space pruning. For example, if the rule set $\{r_1, r_2\}$ (denoted as "12" in the figure) is pruned (Fig. 3b), all its super sets are pruned from the search as well, which greatly reduces the search efforts.

Our *levelwise search* algorithm for finding all minimal inconsistency rule sets is shown in Fig. 4. The function LEVELWISESEARCH takes as inputs a seed rule r_s and a set R of all candidate rules. The algorithm divides the search space (i.e. all subsets of R) into levels, and iteratively processes each level.

The variable L_l stores the subsets of R at the l-th level. For each subset R_l in L_l, the algorithm checks whether R_l is minimal inconsistent (line 6) using Theorem 3. According



(a) Example Lattice



(b) Example Pruned Lattice

Figure 3: Example Set Containment Lattice

```
1:  function LevelwiseSearch($r_s$, $R$)
2:      $l \leftarrow 1$
3:      $L_l \leftarrow \{\{r\}|r \in R\}$
4:      while $L_l \neq \emptyset$ do
5:          for $R_l \in L_l$ do
6:              if $R_l$ is minimal inconsistent then
7:                  report $R_l$ and remove $R_l$ from $L_l$
8:              else if $R_l$ is prunable then
9:                  remove $R_l$ from $L_l$
10:         $L_{l+1} \leftarrow$GenerateNextLevel($L_l$)
11:         $l \leftarrow l + 1$
```

Figure 4: Pseudo-code for Levelwise Search

to Definition 4.9, if R_l is a minimal inconsistent set, all its supersets cannot again be minimal and are thus pruned from the search. And note that the minimality of R_l is automatically guaranteed by the nature of levelwise search, i.e., if any subset $R'_l \subseteq R_l$ is minimal inconsistent, then R'_l (and all its supersets) would be pruned. Otherwise, the algorithm checks if R_l is prunable (line 8) using Corollary 1, i.e., whether R_l and all its supersets cannot be minimal inconsistent. If so, R_l is then pruned from L_l.

After above two procedures, the supersets of the remaining rule sets in L_l are unknown for being minimal inconsistent or not. The algorithm then continues its search in the next level. Following the levelwise search framework, L_{l+1} is initialized as a set (using GenerateNextLevel at line 10) as follows:

$$L_{l+1} = \{R||R| = l+1 \wedge \forall R' \subset R.(|R'| = l \rightarrow R' \in L_l)\}.$$

5. Implementation and Evaluation

In this section, we report our implementation of PSpec. We discuss a case study on TPC-DS benchmark [21] and experiments on synthetic rules.

5.1 Implementation

We have implemented PSpec with XML. Specifically, we defined an XML schema definition (XSD) file, and implemented an XML parser with Java to parse the user-provided XML files. We have also implemented all policy analysis algorithms with Java. We use Z3 SMT solver[1] for policy analysis. To help the data owner write and analyze PSpec specifications, we implemented a GUI-based policy editing tool. Our implementation is available on github[2].

Table 1: Supported Desensitize Operations

Data Category	Supported Desensitize Operations
All	count
Zip	truncate
Income	range
Vehicle	isZero
Price	sum, avg, min, max

5.2 Case Study on TPC-DS

TPC-DS benchmark is a general benchmark for decision support systems. It models a national-wide retail company. The benchmark includes a database schema (7 fact tables and 17 dimension tables) and 99 queries. Some TPC-DS tables contain customers' private information, such as *Customer*, *Customer_demographics*. To ensure these privacy-related data are properly used, we have made a sample PSpec policy. In the case study, we focus on privacy aspects and possible violations in TPC-DS queries.

5.2.1 Vocabulary

We assume only one user category named *Analyst*. The data hierarchy is shown in Fig. 5. Most data categories are self-explanatory. We use abbreviations when meanings are clear. Similar to previous work [17, 28], privacy-related information is classified as *KA* (Key Attribute), *QI* (Quasi Identifier) and *SA* (Sensitive Attribute). They are further divided into finer-grained categories corresponding to columns in the TPC-DS tables.

Table 1 shows supported desensitize operations. There are only a handful of desensitize operations since the schema of TPC-DS is well designed. PSpec rules can directly refer these data categories without desensitize operations.

5.2.2 PSpec Rules

We write PSpec rules to ensure privacy related data is properly used while keeping the data still useful to data analysis (Table 2). The general goal of these rules is to prevent *Analyst* from accessing and projecting private information for individuals. The first rule forbids the access of *KA* since *KA* can directly identify individuals. The rules 2 to 6 forbid the projection of certain data categories since these data categories may identify individuals by linking with public datasets [28]. The rules 7 to 9 further forbid any access to certain combinations of data categories since these combinations have a good chance to identify individuals. Finally, the rules 10 to 13 require some data categories must be desensi-

[1] http://z3.codeplex.com/

[2] Link to repository removed for double blind review.

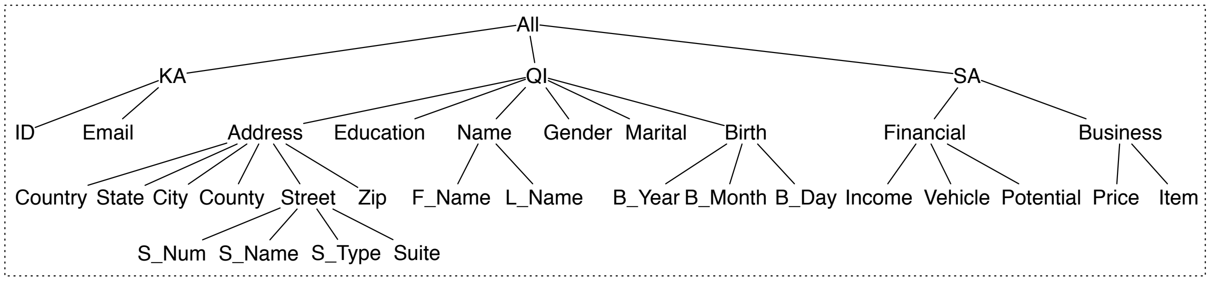


Figure 5: Data Hierarchy for TPC-DS

Table 2: PSpec rules for TPC-DS

ID	Rule
1	`Analyst,[access KA]=>forbid`
2	`Analyst,[projection Street]=>forbid`
3	`Analyst,[projection Name]=>forbid`
4	`Analyst,[projection Birth,projection Address, projection Gender]=>forbid`
5	`Analyst,[projection Birth,projection Address, projection Marital]=>forbid`
6	`Analyst,[projection Birth,projection Address, projection Education]=>forbid`
7	`Analyst,[access F_Name,access L_Name]=>forbid`
8	`Analyst,[access S_Num,access S_Name, access Suite]=>forbid`
9	`Analyst,[access B_Day,access B_Month, access B_Year]=>forbid`
10	`Analyst,[access Zip]=>[{truncate}]`
11	`Analyst,[access QI,access Vehicle]=>[{},{isZero}]`
12	`Analyst,[access QI,access Income]=>[{},{range}]`
13	`Analyst,[access QI,access Price] =>[{},{sum,count,avg,min,max}]`

tized. For example, only aggregated sales price or truncated Zip codes can be used.

5.3 Experimental Evaluation

To evaluate the performance of the policy analysis algorithms, we performed experiments on synthetic rules. All experiments are performed on a laptop with a 2.7 GHZ Intel Core i5 CPU and 8GB of RAM. The maximum memory of JVM is set to 4GB. Each experiment is performed 3 times. The average time is reported.

The policies used in the experiments are randomly generated by a policy generator as follows. We first randomly generate a vocabulary containing N_u user categories and N_d data categories. Both of them are organized as three-level hierarchies. All data categories support the same set of desensitize operations of size N_{ds}. We then randomly generate N_r rules. For each rule, we randomly pick up a user category, n_d data categories ($1 \leq n_d \leq Max_\tau$), and n_{res} restrictions ($1 \leq n_{res} \leq Max_{res}$). Default values for these parameters are listed in Table 3.

We designed several sets of experiments to evaluate the performance of our policy analysis algorithms under different settings. In each set of experiment, only one parameter varies and others take the default values. The experimental results are shown in Table 4.

Table 3: Experiment Parameters

Parameter	Description	Default
N_u	the number of user categories	15
N_d	the number of data categories	60
N_r	the number of rules	600
N_{ds}	the number of desensitize operations	15
Max_τ	the maximum size of data association	5
Max_{res}	the maximum number of restrictions	5

Rules. In the first set of experiments, we change the number of rules N_r to evaluate the basic performance. The number of rules has a direct impact on the performance of both algorithms. The running time for consistency analysis grows more rapidly since its complexity is exponential in the number of rules. The results also show all algorithms finish in reasonable time and hence fit for practical use.

User and Data Categories. In the following two sets of experiments, we change N_u and N_d to evaluate the impact of rule overlaps. Since rules are randomly generated, fewer categories imply more scope overlaps among the randomly generated rules. Rule overlaps have a significant impact on consistency analysis since its efficiency heavily relies on search space pruning. For redundancy analysis, the impact is relatively small since its worst-case time complexity is much lower than consistency analysis.

Data Associations. In the fourth set of experiments, we change Max_τ to evaluate the impact of data associations. Recall that the size of data association in each rule is randomly chosen from 1 to Max_τ.

Increasing Max_τ directly affects redundancy analysis since the time for both scope inclusion test and satisfiability solving increases. For consistency analysis, the time decreases significantly with the increase of Max_τ since longer data associations imply fewer candidate rules for each seed rule on average.

Desensitize Operations. In the fifth set of experiments, we change the number of desensitize operations N_{ds}. Increasing the number of desensitize operations has a slight impact on both algorithms since formulas become more complex and the time for satisfiability solving increases.

Table 4: Experimental Results

Experiment	Parameter	Redundancy	Consistency	Experiment	Parameter	Redundancy	Consistency
Rules	N_r=200	878ms	298ms	Data Associations	Max_τ=1	40ms	48,018ms
	N_r=400	1,237ms	1,155ms		Max_τ=3	2,132ms	8,772ms
	N_r=600	1,690ms	3,914ms		Max_τ=5	2,564ms	3,004ms
	N_r=800	2,445ms	3,467ms		Max_τ=7	5,680ms	1,557ms
	N_r=1000	4,646ms	14,215ms		Max_τ=9	4,560ms	2,243ms
User Categories	N_u=5	2,607ms	14,731ms	Desensitize Operations	N_{ds}=5	1,594ms	2,923ms
	N_u=10	2,067ms	4,046ms		N_{ds}=10	2,603ms	2,586ms
	N_u=15	2,113ms	4,596ms		N_{ds}=15	2,770ms	4,755ms
	N_u=20	2,043ms	1,726ms		N_{ds}=20	3,983ms	5,421ms
	N_u=25	2,426ms	1,923ms		N_{ds}=25	2,296ms	4,697ms
Data Categories	N_d=20	2,628ms	7,597ms	Restrictions	Max_{res}=1	1,593ms	2,515ms
	N_d=40	4,348ms	4,931ms		Max_{res}=3	1,109ms	2,692ms
	N_d=60	3,015ms	3,269ms		Max_{res}=5	2,533ms	3,831ms
	N_d=80	5,763ms	2,127ms		Max_{res}=7	3,627ms	3,115ms
	N_d=100	1,980ms	1,715ms		Max_{res}=9	5,787ms	3,732ms

Restrictions. In the final set of experiments, we change Max_{res} to evaluate the impact of restrictions. Similar to the previous set, increasing the number of restrictions slightly impacts the running time of both algorithms since the time needed for satisfiability solving increases.

6. Related Works

Privacy Languages. Several privacy languages have been proposed to formalize text-based policies, including P3P [7], EPAL [3], XACML [19], and P-RBAC models [23]. Several works have also been proposed to formalize privacy laws, including pLogic [15], contextual integrity [5], and PrivacyLFP [8]. These works serve for general purposes. PSpec is designed as a specific language suitable for big data analytics systems with explicitly support for data association and desensitization.

Sen et al. [27] recently proposed LEGALEASE, a privacy language for encoding privacy policies in big data systems. But LEGALEASE is designed for handling general data usages, that is, store, use and share. It may not be most suitable for online data analytics. For instance, it has no flexible support for specifying desensitizations. Moreover, the work [27] lacks necessary policy analysis algorithms for redundancy and consistency. Such algorithms are useful to analyze unintended effects after a policy is drafted.

Policy Analysis. Policy analysis has been extensively studied in past decades. These works range from system configuration policies [1], firewall policies [2, 14], to access control policies [13, 16]. Although the intuition of redundancy and consistency is similar, the policy analysis algorithms in this paper are pertaining to PSpec. They are further complicated by data association.

Privacy Extensions for Programming Languages. In order to automatically enforce privacy requirements on programs, several extensions for programming languages have been proposed, including Jeeves [29] for enforcing privacy policies, and Fuzz [24] and DFuzz [11] for certifying differentially private [9] programs. There are many language extensions to enforce security policies, such as Jif [20], Joana [12], and FINE [6]. Different from these works, PSpec is a new privacy language for data analytics. It is not an extension to existing programming languages.

7. Conclusion

In this paper, we present the privacy language PSpec for specifying data usage restrictions in data analytics. To facilitate the user reason and analyze the PSpec policy, we also introduce two policy analysis algorithms, namely redundancy analysis and conflict analysis.

In the future, we plan to perform more real world case studies to further evaluate the applicability of PSpec. Moreover, we also plan to explore other policy analysis algorithms to help users write and analyze their policies.

References

[1] D. Agrawal, J. Giles, K.-W. Lee, and J. Lobo. Policy ratification. In *Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on*, pages 223–232. IEEE, 2005.

[2] E. S. Al-Shaer and H. H. Hamed. Discovery of policy anomalies in distributed firewalls. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2605–2616. IEEE, 2004.

[3] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (epal 1.2). *Submission to W3C*, 2003.

[4] M. Barbaro and J. T. Zeller. A face is exposed for AOL searcher no. 4417749. New York Times, August 2006.

[5] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

[6] J. Chen, R. Chugh, and N. Swamy. Type-preserving compilation of end-to-end verification of security enforcement. In *ACM Sigplan Notices*, volume 45, pages 412–423. ACM, 2010.

[7] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. The platform for privacy preferences 1.0 (p3p1. 0) specification. *W3C recommendation*, 16, 2002.

[8] H. DeYoung, D. Garg, L. Jia, D. Kaynar, and A. Datta. Experiences in the logical specification of the hipaa and glba privacy laws. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, pages 73–82. ACM, 2010.

[9] C. Dwork. Differential privacy. In *Automata, languages and programming*, pages 1–12. Springer, 2006.

[10] C. Dwork. Differential privacy: A survey of results. In *Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[11] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce. Linear dependent types for differential privacy. In *ACM SIGPLAN Notices*, volume 48, pages 357–370. ACM, 2013.

[12] C. Hammer and G. Snelting. Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs. *International Journal of Information Security*, 8(6):399–422, 2009.

[13] H. Hu, G.-J. Ahn, and K. Kulkarni. Anomaly discovery and resolution in web access control policies. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 165–174. ACM, 2011.

[14] H. Hu, G.-J. Ahn, and K. Kulkarni. Detecting and resolving firewall policy anomalies. *Dependable and Secure Computing, IEEE Transactions on*, 9(3):318–331, 2012.

[15] P. E. Lam, J. C. Mitchell, and S. Sundaram. *A formalization of HIPAA for a medical messaging system*. Springer, 2009.

[16] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. Exam: a comprehensive environment for the analysis of access control policies. *International Journal of Information Security*, 9(4): 253–273, 2010.

[17] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1 (1):3, 2007.

[18] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data mining and knowledge discovery*, 1(3):241–258, 1997.

[19] T. Moses et al. Extensible access control markup language (xacml) version 2.0. *Oasis Standard*, 200502, 2005.

[20] A. C. Myers. Jflow: Practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 228–241. ACM, 1999.

[21] R. O. Nambiar and M. Poess. The making of TPC-DS. In *Proceedings of the 32nd international conference on Very large data bases*, pages 1049–1058. VLDB Endowment, 2006.

[22] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the 29th IEEE Symposium on Security & Privacy*, 2008.

[23] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C.-M. Karat, J. Karat, and A. Trombeta. Privacy-aware role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 13(3):24, 2010.

[24] J. Reed and B. C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. *ACM Sigplan Notices*, 45(9):157–168, 2010.

[25] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *Selected Areas in Communications, IEEE Journal on*, 21(1):5–19, 2003.

[26] A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *Computer Security Foundations, 2005. CSFW-18 2005. 18th IEEE Workshop*, pages 255–269. IEEE, 2005.

[27] S. Sen, S. Guha, A. Datta, S. K. Rajamani, J. Tsai, and J. M. Wing. Bootstrapping privacy compliance in big data systems. In *Proceedings of the 35th IEEE Symposium on Security & Privacy (Oakland)*, 2014.

[28] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[29] J. Yang, K. Yessenov, and A. Solar-Lezama. A language for automatically enforcing privacy policies. In *ACM SIGPLAN Notices*, volume 47, pages 85–96. ACM, 2012.