Strong and Provably Secure Database Access Control

Marco Guarnieri
Institute of Information Security
Department of Computer Science
ETH Zurich, Switzerland
marco.guarnieri@inf.ethz.ch

Srdjan Marinovic
The Wireless Registry, Inc.
Washington DC, US
srdjan@wirelessregistry.com

David Basin
Institute of Information Security
Department of Computer Science
ETH Zurich, Switzerland
basin@inf.ethz.ch

ABSTRACT

Existing SQL access control mechanisms are extremely limited. Attackers can leak information and escalate their privileges using advanced database features such as views, triggers, and integrity constraints. This is not merely a problem of vendors lagging behind the state-of-the-art. The theoretical foundations for database security lack adequate security definitions and a realistic attacker model, both of which are needed to evaluate the security of modern databases. We address these issues and present a provably secure access control mechanism that prevents attacks that defeat popular SQL database systems.

1. INTRODUCTION

It is essential to control access to databases that store sensitive information. To this end, the SQL standard defines access control rules and all SQL database vendors have accordingly developed access control mechanisms. The standard however fails to define a precise access control semantics, the attacker model, and the security properties that the mechanisms ought to satisfy. As a consequence, existing access control mechanisms are implemented in an ad hoc fashion, with neither precise security guarantees nor the means to verify them.

This deficit has dire and immediate consequences. We show that popular database systems are susceptible to two types of attacks. Integrity attacks allow an attacker to perform non-authorized changes to the database. Confidentiality attacks allow an attacker to learn sensitive data. These attacks exploit advanced SQL features, such as triggers, views, and integrity constraints, and they are easy to carry out.

Current research efforts in database security are neither adequate for evaluating the security of modern databases, nor do they account for their advanced features. In more detail, existing research [4, 12, 35, 46] implicitly considers attackers who use `SELECT` commands. But the capabilities offered by databases go far beyond `SELECT`. Users, in general, can modify the database's state and security policy, as well as use features such as triggers, views, and integrity constraints. Consequently, all proposed research solutions fail to prevent attacks such as those we present in §2.

In summary, the database vendors have been left to develop access control mechanisms without guidance from either the SQL standard or existing research in database security. It is therefore not surprising that modern databases are open to abuse.

Contributions. We develop a comprehensive formal framework for the design and analysis of database access control. We use it to design and verify an access control mechanism that prevents confidentiality and integrity attacks that defeat existing mechanisms.

First, we develop an operational semantics for databases that supports SQL's core features, as well as triggers, views, and integrity constraints. Our semantics models both the security-critical aspects of these features and the database's dynamic behaviour at the level needed to capture realistic attacks. Our semantics is substantially more detailed than those used in previous works [35, 46], which ignore the database's dynamics.

Second, we develop a novel attacker model that, in addition to SQL's core features, incorporates advanced features such as triggers, views, and integrity constraints. Furthermore, our attacker can infer information based on the semantics of these features. Note that our attacker model subsumes the `SELECT`-only attacker considered in previous works [35], [46]. We also develop an executable version of our operational semantics and attacker model using the Maude term-rewriting framework [14]. The executable model acts as a reference implementation for our semantics. Given the complexity of databases and their features, having an executable version of our models provides a way to validate them against existing database systems and against the examples we use in this paper.

Third, we present two security definitions—database integrity and data confidentiality—that reflect two principal security requirements for database access control. There is a natural and intuitive relationship between these definitions and the types of attacks that we identify. We thus argue that these definitions provide a strong measure of whether a given access control mechanism prevents our attacker from exploiting modern SQL databases.

Finally, using our framework, we build a database access control mechanism that is provably secure with respect to our attacker model and security definitions. In contrast to existing mechanisms, our solution prevents all the attacks that we report on in §2.

Related Work. Surprisingly, and in contrast to other areas of information security [19], there does not exist a well-defined attacker model for database access control. From the literature, we extracted the `SELECT`-only attacker model, where the attacker uses just `SELECT` commands. A number of access control mechanisms, such as [1, 4, 8, 9, 13, 27, 31, 35, 41, 43, 46], implicitly consider this attacker model. The boundaries of this model are blurred and the attacker's capabilities are unclear. For instance, only a few works, such as [46], explicitly state that update commands are not supported, whereas others [4, 8, 9, 35] ignore what the attacker can learn from update commands. Works on Inference Control [12, 20, 44] and Controlled Query Evaluation [11] consider a variation of the `SELECT`-only attacker, in which the attacker additionally has some initial knowledge about the data and can derive new information from the query's results through inference rules. Note that while [44] supports update commands, it treats them just as a way of increasing data availability, rather than considering them as a possible attack vector.

Database access control mechanisms can be classified into two distinct families [35]. Mechanisms in the *Truman model* [4,46] transparently modify query results to restrict the user's access to the data authorized by the policy. In contrast, mechanisms in the *Non-Truman model* [8, 9, 35] either accept or reject queries without modifying their results. Different notions of security have been proposed for these models [24, 35, 46]. They are, however, based on `SELECT`-only attackers and provide no security guarantees against realistic attackers that can alter the database and the policy or use advanced SQL features. We refer the reader to §7 for further comparison with related work.

Organization. In §2 we present attacks that illustrate serious weaknesses in existing Database Management Systems (DBMSs). In §3 we introduce background and notation about queries, views, triggers, and access control. In §4 we formalize our system and attacker models, and in §5 we define the desired security properties. In §6 we present our access control mechanism, and in §7 we discuss related work. Finally, we draw conclusions in §8. The system's operational semantics, the attacker model, and complete proofs of all results are in Appendices A–H. A prototype of our enforcement mechanism and its executable semantics are available at [26]. This technical report is an extended version of [25].

2. ILLUSTRATIVE ATTACKS

We demonstrate here how attackers can exploit existing DBMSs using standard SQL features. We classify these attacks as either *Integrity Attacks* or *Confidentiality Attacks*. In the former, an attacker makes unauthorized changes to the database, which stores the data, the policy, the triggers, and the views. In the latter, an attacker learns sensitive data by interacting with the system and observing the outcome. No existing access control mechanism prevents all the attacks we present. Moreover, many related attacks can be constructed using variants of the ideas presented here. We manually carried out the attacks against IBM DB2, Oracle Database, PostgreSQL, MySQL, SQL Server, and Firebird. We summarize our findings at the end of this section.

2.1 Integrity Attacks

Our three integrity attacks combine different database features: `INSERT`, `DELETE`, `GRANT`, and `REVOKE` commands together with views and triggers. In the first attack, an attacker creates a trigger, i.e., a procedure automatically executed by the DBMS in response to user commands, that will be activated by an unaware user with a higher security clearance and will perform unauthorized changes to the database. The attack requires triggers to be executed under the privileges of the users activating them. Such triggers are supported by PostgreSQL, SQL Server, and Firebird.

Attack 1. **Triggers with activator's privileges.** Consider a database with two tables P and S and two users u_1 and u_2. The attacker is the user u_1, whose goal is to delete the content of S. The policy is that u_1 is not authorized[1] to alter S, u_1 can create triggers on P, and u_2 can read and modify S and P. The attack is as follows:

1. u_1 creates the trigger:

```
CREATE TRIGGER t ON P AFTER INSERT
  DELETE FROM S;
```

2. u_1 waits until u_2 inserts a tuple into the table P. The trigger will then be invoked using u_2's privileges and S's content will be deleted. ■

An attacker can use similar attacks to execute arbitrary commands with administrative privileges. Despite the threat posed by such simple attacks, the existing countermeasures [2] are unsatisfactory; they are either too restrictive, for instance completely disabling triggers in the database, or too time consuming and error prone, namely manually checking if "dangerous" triggers have been created.

In our second attack, an attacker escalates his privileges by delegating the read permission for a table without being authorized to delegate this permission. The attacker first creates a view over the table and, afterwards, delegates the access to the view to another user. This attack exploits DBMSs, such as PostgreSQL, where a user can grant any read permission over his own views. Note that `GRANT` and `REVOKE` commands are *write operations*, which target the database's internal configuration instead of the tables.

Attack 2. **Granting views.** Consider a database with a table S, two users u_1 and u_2, and the following policy: u_1 can create views and read S (without being able to delegate this permissions), and u_2 cannot read S. The attack is as follows:

1. u_1 creates the view: `CREATE VIEW` v `AS SELECT` $*$ `FROM` S.
2. u_1 issues the command `GRANT SELECT ON` v `TO` u_2. Now, u_2 can read S through v. However, u_1 is not authorized to delegate the read permission on S. ■

This attack exploits several subtleties in the commands' semantics: (a) users can create views over all tables they can read, (b) the views are executed under the owner's privileges, and (c) view's owners can grant arbitrary permissions over their own views. These features give u_1 the implicit ability to delegate the read access over S. As a result, the overall system's behaviour does not conform with the given policy. That is, u_1 should not be permitted to delegate the read access to S or to any view that depends on it. Note that the commands' semantics may vary between different DBMSs.

In our third attack, an attacker exploits the failure of access control mechanisms to propagate `REVOKE` commands.

Attack 3. **Revoking views.** Consider a database with a table S, three users u_1, u_2, and u_3, and the following policy: u_1 can read S and delegate this permission, u_2 can create views, and u_3 cannot read S. The attack proceeds as follows:

1. u_1 issues the command `GRANT SELECT ON` S `TO` u_2 `WITH GRANT OPTION`.
2. u_2 creates the view: `CREATE VIEW` v `AS SELECT` $*$ `FROM` S.
3. u_2 issues the command `GRANT SELECT ON` v `TO` u_3.

[1] As is common in SQL, a user is authorized to execute a command if and only if the policy assigns him the corresponding permission.

DBMS	Integrity Attacks			Confidentiality Attacks	
	Triggers with activator's privileges	Granting views	Revoking views	Table updates and integrity constraints	Triggers with owner's privileges
IBM DB2 (v. 10.5)	†	$\mathcal{X}$	✓	✓	✓
Oracle (v. 11g)	†	$\mathcal{X}$	$\mathcal{X}$	✓	✓
PostgreSQL (v. 9.3.5)	✓	✓	✓	✓	✓
MySQL (v. 14.14)	†	$\mathcal{X}$	✓	✓	✓
SQL Server (v. 12.0)	✓	†	†	✓	✓
Firebird (v. 2.5.2)	✓	$\mathcal{X}$	✓	✓	✓

Figure 1: The ✓ symbol indicates a successful attack, whereas $\mathcal{X}$ indicates a failed attack. The † symbol indicates that the DBMS does not support the features necessary to launch the attack.

4. u_1 revokes the permission to read S (and to delegate the permission) from u_2: `REVOKE SELECT ON` S `FROM` u_2. Now, u_3 cannot read v because u_2, which is v's owner, cannot read S.
5. u_1 grants again the permission to read S to u_2: `GRANT SELECT ON` S `TO` u_2. Now, u_3 can again read v but u_2 can no longer delegate the read permission on v. ■

This attack succeeds because, in the fourth step, the `REVOKE` statement does not remove the `GRANT` granted by u_2 to u_3 to read v. This `GRANT` only becomes ineffective because u_2 is no longer authorized to read S. However, after the fifth step, this `GRANT` becomes effective again, even though u_2 can no longer delegate the read permission on v. Thus, the policy is left in an inconsistent state.

2.2 Confidentiality Attacks

We now present two attacks that use `INSERT` and `SELECT` commands together with triggers and integrity constraints. In our fourth attack, an attacker exploits integrity constraint violations to learn sensitive information. An integrity constraint is an invariant that must be satisfied for a database state to be considered *valid*. *Integrity constraint violations* arise when the execution of an SQL command leads the database from a valid state into an invalid one.

Attack 4. **Table updates and integrity constraints.** Consider a database with two tables P and S. Suppose the primary key of both tables is the user's identifier. Furthermore, the set of user identifiers in S is contained in the set of user identifiers in P, i.e., there is a foreign key from S to P. The attacker is the user u whose goal is to learn whether `Bob` is in S. The access control policy is that u can read P and insert tuples in S. The attacker u can learn whether `Bob` is in S as follows:
1. He reads P and learns `Bob`'s identifier.
2. He issues an `INSERT` statement in S using `Bob`'s id.
3. If `Bob` is already in S, then u gets an error message about the primary key's violation. Alternatively, there is no violation and u learns that `Bob` is not in S. ■

Even though similar attacks have been identified before [29, 40], existing DBMSs are still vulnerable.

In our fifth attack, an attacker learns sensitive information by exploiting the system's triggers. The trigger in this attack is executed under the privileges of the trigger's owner. Such triggers are supported by IBM DB2, Oracle Database, PostgreSQL, MySQL, SQL Server, and Firebird.

Attack 5. **Triggers with owner's privileges.** Consider a database with three tables N, P, and T. The attacker is the user u, who wishes to learn whether v is in T. The policy is that u is not authorized to read the table T, and he can read and modify the tables N and P. Moreover, the following trigger has been defined by the administrator.

```
CREATE TRIGGER t ON P AFTER INSERT FOR EACH ROW
  IF exists(SELECT * FROM T WHERE id = NEW.id)
   INSERT INTO N VALUES (NEW.id);
```

The attack is as follows:
1. u deletes v from N.
2. u issues the command `INSERT INTO` P `VALUES (`v`)`.
3. u checks the table N. If it contains v's id, then v is in T. Otherwise, v is not in T. ■

This attack exploits that the trigger t conditionally modifies the database. Furthermore, the attacker can activate t, by inserting tuples in P, and then observe t's effects, by reading the table N. He therefore can exploit t's execution to learn whether t's condition holds. We assume here that the attacker knows the triggers in the system. This is, in general, a weak assumption as triggers usually describe the domain-specific rules regulating a system's behaviour and users are usually aware of them.

2.3 Discussion

We manually carried out all five attacks against IBM DB2, Oracle Database, PostgreSQL, MySQL, SQL Server, and Firebird. Figure 1 summarizes our findings. None of these systems prevent the confidentiality attacks. They are however more successful in preventing the integrity attacks. The most successful is Oracle Database, which prevents two of the three attacks, while Attack 1 cannot be carried out due to missing features. IBM DB2, MySQL, and Firebird prevent just one of the three attacks, namely Attack 2. However, they all fail to prevent Attack 3. Note that Firebird also fails to prevent Attack 1. In contrast, Attack 1 cannot be carried out against MySQL and IBM DB2 due to missing features. SQL Server also fails to prevent Attack 1; however the remaining two attacks cannot be carried out due to missing features. PostgreSQL fails to prevent all three attacks.

We argue that the dire state of database access control mechanisms, as illustrated by these attacks, comes from the lack of clearly defined security properties that such mechanisms ought to satisfy and the lack of a well-defined attacker model. We therefore develop a formal attacker model and precise security properties and we use them to design a provably secure access control mechanism that prevents all the above attacks.

3. DATABASE MODEL

We now formalize databases including features like views, access control policies, and triggers. Our formalization of databases and queries follows [3], and our access control policies formalize SQL policies.

3.1 Overview

In this paper we consider the following SQL features: `SELECT`, `INSERT`, `DELETE`, `GRANT`, `REVOKE`, `CREATE TRIGGER`, `CREATE VIEW`, and `ADD USER` commands.

For `SELECT` commands, rather than using SQL, we use the relational calculus (RC), i.e., function-free first-order logic, which has a simple and well-defined semantics [3]. We support `GRANT` commands with the `GRANT OPTION` and `REVOKE` commands with the `CASCADE OPTION`, i.e., when a user revokes a privilege, he also revokes all the privileges that depend on it. We support `INSERT` and `DELETE` commands that explicitly identify the tuple to be inserted or deleted, i.e., commands of the form `INSERT INTO` *table*$(x_1, \ldots, x_n)$ `VALUES` $(v_1, \ldots, v_n)$ and `DELETE FROM` *table* `WHERE` $x_1 = v_1 \wedge \ldots \wedge x_n = v_n$, where $x_1, \ldots, x_n$ are *table*'s attributes and $v_1, \ldots, v_n$ are the tuple's values. More complex `INSERT` and `DELETE` commands, as well as `UPDATE`s, can be simulated by combining `SELECT`, `INSERT`, and `DELETE` commands.

We support only `AFTER` triggers on `INSERT` and `DELETE` events, i.e., triggers that are executed in response to `INSERT` and `DELETE` commands. The triggers' `WHEN` conditions are arbitrary boolean queries and their actions are `GRANT`, `REVOKE`, `INSERT`, or `DELETE` commands. Note that DBMSs usually impose severe restrictions on the `WHEN` clause, such as it must not contain sub-queries. However, most DBMSs can express arbitrary conditions on triggers by combining control flow statements with `SELECT` commands inside the trigger's body. Thus, we support the class of triggers whose body is of the form `BEGIN IF` *expr* `THEN` *act* `END`, where *act* is either a `GRANT`, `REVOKE`, `INSERT`, or `DELETE` command. Note that all triggers used in §2 belong to this class.

We support two kinds of integrity constraints: functional dependencies and inclusion dependencies [3]. They model the most widely used families of SQL integrity constraints, namely the `UNIQUE`, `PRIMARY KEY`, and `FOREIGN KEY` constraints. We also support views with both the owner's privileges and the activator's privileges.

The SQL fragment we support, shown in Figure 41, contains the most common SQL commands for data manipulation and access control as well as the core commands for creating triggers and views. The ideas and the techniques presented in this paper are general and can be extended to the entire SQL standard.

3.2 Databases and Queries

Let $\mathcal{R}$, $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{T}$ be mutually disjoint, countably infinite sets, respectively representing identifiers of relation schemas, users, views, and triggers.

A *database schema* D is a pair $\langle \Sigma, \mathbf{dom} \rangle$, where Σ is a first-order signature and $\mathbf{dom}$ is a fixed countably infinite domain. The signature Σ consists of a set of *relation schemas* $R \in \mathcal{R}$, also called *tables*, with arity $|R|$ and sort $sort(R)$. A *state* s of D is a finite Σ-structure over $\mathbf{dom}$. We denote by Ω_D the set of all states. Given a table $R \in D$, $s(R)$ denotes the set of tuples that belong to R in s.

A *query* q over a schema D is of the form $\{\overline{x} \mid \phi\}$, where $\overline{x}$ is a sequence of variables, ϕ is a relational calculus formula over D, and ϕ's free variables are those in $\overline{x}$. A *boolean query* is a query $\{ \mid \phi\}$, also written as ϕ, where ϕ is a sentence. The result of executing a query q on a state s, denoted by $[q]^s$, is a boolean value in $\{\top, \bot\}$, if q is a boolean query, or a set of tuples otherwise. We denote by RC (respectively RC_{bool}) the set of all relational calculus queries (respectively sentences). We consider only *domain-independent queries* as is standard, and we employ the standard relational calculus semantics [3].

Let $D = \langle \Sigma, \mathbf{dom} \rangle$ be a schema, s be a state in Ω_D, R be a table in D, and $\bar{t}$ be a tuple in $\mathbf{dom}^{|R|}$. The result of inserting (respectively deleting) $\bar{t}$ in R in the state s is the state s', denoted by $s[R \oplus \bar{t}]$ (respectively $s[R \ominus \bar{t}]$), where $s'(T) = s(T)$ for all $T \in \Sigma$ such that $T \neq R$, and $s'(R) = s(R) \cup \{\bar{t}\}$ (respectively $s'(R) = s(R) \setminus \{\bar{t}\}$).

An *integrity constraint over* D is a relational calculus sentence γ over D. Given a state s, we say that *s satisfies the constraint γ* iff $[\gamma]^s = \top$. Given a set of constraints Γ, Ω_D^Γ denotes the set of all states satisfying the constraints in Γ, i.e., $\Omega_D^\Gamma = \{s \in \Omega_D \mid \bigwedge_{\gamma \in \Gamma} [\gamma]^s = \top\}$. We consider two types of integrity constraints: *functional dependencies*, which are sentences of the form $\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. ((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}')$, and *inclusion dependencies*, which are sentence of the form $\forall \overline{x}, \overline{y}. (R(\overline{x}, \overline{y}) \Rightarrow \exists \overline{z}. S(\overline{x}, \overline{z}))$.

3.3 Views

Let D be a schema. A *view V over D* is a tuple $\langle id, o, q, m \rangle$, where $id \in \mathcal{V}$ is the view identifier, $o \in \mathcal{U}$ is the view's owner, q is the non-boolean query over D defining the view, and $m \in \{A, O\}$ is the security mode, where A stands for *activator's privileges* and O stands for *owner's privileges*. Note that the query q may refer to other views. We assume, however, that views have no cyclic dependencies between them. We denote by $\mathcal{VIEW}_D$ the set of all views over D. The *materialization of a view $\langle V, o, q, m \rangle$ in a state s*, denoted by $s(V)$, is $[q]^s$. We extend the relational calculus in the standard way to work with views [3].

3.4 Access Control Policies

We now formalize the SQL access control model. We first formalize five privileges. Let D be a database schema. A *`SELECT` privilege over D* is a tuple $\langle$`SELECT`$, R\rangle$, where R is a relation schema in D or a view over D. A *`CREATE VIEW` privilege over D* is a tuple $\langle$`CREATE VIEW`$\rangle$. An *`INSERT` privilege over D* is a tuple $\langle$`INSERT`$, R\rangle$, a *`DELETE` privilege over D* is a tuple $\langle$`DELETE`$, R\rangle$, and a *`CREATE TRIGGER` privilege over D* is a tuple $\langle$`CREATE TRIGGER`$, R\rangle$, where R is a relation schema in D. We denote by $\mathcal{PRIV}_D$ the set of privileges over D.

Following SQL, we use `GRANT` commands to assign privileges to users. Let $U \subseteq \mathcal{U}$ be a set of users and D be a database schema. We now define (U, D)-grants and (U, D)-revokes. There are two types of (U, D)-grants. A *(U, D)-simple grant* is a tuple $\langle \oplus, u, p, u' \rangle$, where $u \in U$ is the user receiving the privilege $p \in \mathcal{PRIV}_D$ and $u' \in U$ is the user granting this privilege. A *(U, D)-grant with grant option* is a tuple $\langle \oplus^*, u, p, u' \rangle$, where u, p, and u' are as before. A *(U, D)-revoke* is a tuple $\langle \ominus, u, p, u' \rangle$, where $u \in U$ is the user from which the privilege $p \in \mathcal{PRIV}_D$ will be revoked and $u' \in U$ is the user revoking this privilege. We denote by $\Omega_{U,D}^{sec}$ the set of all (U, D)-grants and (U, D)-revokes. A grant $\langle \oplus, u, p, u' \rangle$ models the command `GRANT` p `TO` u issued by u', a grant with grant option $\langle \oplus^*, u, p, u' \rangle$ models the command `GRANT` p `TO` u `WITH GRANT OPTION` issued by u', and a revoke

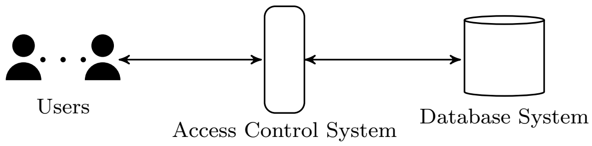


Figure 2: System model.

$\langle \ominus, u, p, u' \rangle$ models the command `REVOKE` p `FROM` u `CASCADE` issued by u'.

Finally, we define a (U, D)-*access control policy* S as a finite set of (U, D)-grants. We denote by $\mathcal{S}_{U,D}$ the set of all (U, D)-policies.

Example 3.1. Consider the policy described in Attack 5. The database D has three tables: N, P, and T. The set U is $\{u, admin\}$ and the policy S contains the following grants: $\langle \oplus, u, \langle \texttt{SELECT}, P\rangle, admin\rangle$, $\langle \oplus, u, \langle \texttt{INSERT}, P\rangle, admin\rangle$, $\langle \oplus, u, \langle \texttt{DELETE}, P\rangle, admin\rangle$, $\langle \oplus, u, \langle \texttt{SELECT}, N\rangle, admin\rangle$, $\langle \oplus, u, \langle \texttt{INSERT}, N\rangle, admin\rangle$, and $\langle \oplus, u, \langle \texttt{DELETE}, N\rangle, admin\rangle$. ■

3.5 Triggers

Let D be a database schema. A *trigger over* D is a tuple $\langle id, u, e, R, \phi, a, m\rangle$, where $id \in \mathcal{T}$ is the trigger identifier, $u \in \mathcal{U}$ is the trigger's owner, $e \in \{INS, DEL\}$ is the trigger event (where *INS* stands for `INSERT` and *DEL* stands for `DELETE`), $R \in D$ is a relation schema, the trigger condition ϕ is a relational calculus formula such that $free(\phi) \subseteq \{x_1, \ldots, x_{|R|}\}$, and the trigger action a is one of: (1) $\langle \texttt{INSERT}, R', \bar{t}\rangle$, where $R' \in D$ and $\bar{t}$ is a $|R'|$-tuple of values in **dom** and variables in $\{x_1, \ldots, x_{|R|}\}$, (2) $\langle \texttt{DELETE}, R', \bar{t}\rangle$, where R' and $\bar{t}$ are as before, or (3) $\langle op, u, p\rangle$, where $op \in \{\oplus, \oplus^*, \ominus\}$, $u \in \mathcal{U}$, and p is a privilege over D. Finally, $m \in \{A, O\}$ is the security mode, where A stands for *activator's privileges* and O stands for *owner's privileges*. We denote by $\mathcal{TRIGGER}_D$ the set of all triggers over D.

We assume that any command a is executed atomically together with all the triggers activated by a. We also assume that triggers do not recursively activate other triggers. Hence all executions terminate. We enforce this condition syntactically at the trigger's creation time; see Appendix A for additional details. The trigger $\langle t, admin, INS, P, T(x_1), \langle \texttt{INSERT}, N, x_1\rangle, O\rangle$ models the trigger in Attack 5. Here, x_1 is bound, at run-time, to the value inserted in P by the trigger's invoker.

4. SYSTEM AND ATTACKER MODEL

We next present our system and attacker models. Executable versions of these models, built in the Maude framework [14], are available at [26]. The models can be used for simulating the execution of our operational semantics, as well as computing the information that an attacker can infer from the system's behaviour. We have executed and validated all of our examples using these models.

4.1 Overview

In our system model, shown in Figure 2, users interact with two components: a database system and an access control system. The access control system contains both a policy enforcement point and a policy decision point. We assume that all the communication between the users and the components is over secure channels.

Database System. The database system (or database for short) manages the data. The database's state is represented by a mapping from relation schemas to sets of tuples. We assume that all database operations are atomic.

Users. Users interact with the database where each command is checked by the access control system. Each user has a unique account through which he can issue `SELECT`, `INSERT`, `DELETE`, `GRANT`, `REVOKE`, `CREATE TRIGGER`, and `CREATE VIEW` commands.

The *system administrator* is a distinguished user responsible for defining the database schema and the access control policy. In addition to issuing queries and commands, he can create user accounts and assign them to users. The administrator interacts with the access control system through a special account *admin*.

The *attacker* is a user, other than the administrator, with an assigned user account who attempts to violate the access control policy. Namely, his goals are: (1) to read or infer data from the database for which he lacks the necessary `SELECT` privileges, and (2) to alter the system state in unauthorized ways, e.g., changing data in relations for which he lacks the necessary `INSERT` and `DELETE` privileges. The attacker can issue any command available to users and he sees the results of his commands. The attacker's inference capabilities are specified using deduction rules.

Access Control System. The access control system protects the confidentiality and integrity of the data in the database. It is configured with an access control policy S, it intercepts all commands issued by the users, and it prevents the execution of commands that are not authorized by S. When a user u issues a command c, the access control system decides whether u is authorized to execute c. If c complies with the policy, then the access control system forwards the command to the DBMS, which executes c and returns its result to u. Otherwise, it raises a *security exception* and rejects c. Note that this corresponds to the Non-Truman model [35]; see related work for more details.

The access control system also logs all issued commands. When evaluating a command, the access control system can access the database's current state and the log.

4.2 System Model

We formalize our system model as a labelled transition system (LTS). First, we define a system configuration, which describes the database schema and the integrity constraints, and the user actions. Afterwards, we define the system's state, which represents a snapshot of the system that contains the database's state, the identifiers of the users interacting with the system, the access control policy, and the current triggers and views in the system. Finally, we formalize the system's behaviour as a small step operational semantics, including all features necessary to reason about security, even in the presence of attacks like those illustrated in §2.

A *system configuration* is a tuple $\langle D, \Gamma\rangle$ such that D is a schema and Γ is a finite set of integrity constraints over D. Let $M = \langle D, \Gamma\rangle$ be a system configuration and $u \in \mathcal{U}$ be a user. A (D, u)-*action* is one of the following tuples:

- $\langle u, \texttt{ADD\_USER}, u'\rangle$, where $u = admin$ and $u' \in \mathcal{U} \setminus \{admin\}$,
- $\langle u, \texttt{SELECT}, q\rangle$, where q is a boolean query[2] over D,

[2] Without loss of generality, we focus only on boolean queries [3]. We can support non-boolean queries as follows. Given a

- $\langle u, \texttt{INSERT}, R, \overline{t}\rangle$, where $R \in D$ and $\overline{t} \in \mathbf{dom}^{|R|}$,
- $\langle u, \texttt{DELETE}, R, \overline{t}\rangle$, where R and $\overline{t}$ are as above,
- $\langle op, u', p, u\rangle$, where $\langle op, u', p, u\rangle \in \Omega^{sec}_{D,\mathcal{U}}$, or
- $\langle u, \texttt{CREATE}, o\rangle$, where $o \in \mathcal{TRIGGER}_D \cup \mathcal{VIEW}_D$.

We denote by $\mathcal{A}_{D,u}$ the set of all (D, u)-actions and by $\mathcal{A}_{D,U}$, for some $U \subseteq \mathcal{U}$, the set $\bigcup_{u\in U} \mathcal{A}_{D,u}$.

An *M-context* describes the system's history, the scheduled triggers that must be executed, and how to modify the system's state in case a roll-back occurs. We denote by $\mathcal{C}_M$ the set of all M-contexts. We assume that $\mathcal{C}_M$ contains a distinguished element ϵ representing the empty context, which is the context in which the system starts. Contexts are formalized in Appendix A.

An *M-state* is a tuple $\langle db, U, sec, T, V, c\rangle$ such that $db \in \Omega^{\Gamma}_D$ is a database state, $U \subset \mathcal{U}$ is a finite set of users such that $admin \in U$, $sec \in \mathcal{S}_{U,D}$ is a security policy, T is a finite set of triggers over D owned by users in U, V is a finite set of views over D owned by users in U, and $c \in \mathcal{C}_M$ is an M-context. We denote by Ω_M the set of all M-states. An M-state $\langle db, U, sec, T, V, c\rangle$ is *initial* iff (a) sec contains only grants issued by $admin$, (b) T (respectively V) contains only triggers (respectively views) owned by $admin$, and (c) $c = \epsilon$. We denote by $\mathcal{I}_M$ the set of all initial states.

An *M-Policy Decision Point* (M-PDP) is a total function $f : \Omega_M \times \mathcal{A}_{D,\mathcal{U}} \to \{\top, \bot\}$ that maps each state s and action a to an access control decision represented by a boolean value, where $\top$ stands for permit and $\bot$ stands for deny. An *extended configuration* is a tuple $\langle M, f\rangle$, where M is a system configuration and f is an M-PDP.

We now define the LTS representing the system model.

Definition 4.1. Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ and f is an M-PDP. The *P-LTS* is the labelled transition system $\langle S, A, \to_f, I\rangle$ where $S = \Omega_M$ is the set of states, $A = \mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D$ is the set of actions, $\to_f \subseteq S \times A \times S$ is the transition relation, and $I = \mathcal{I}_M$ is the set of initial states. □

Let $P = \langle M, f\rangle$ be an extended configuration. A *run* r of a P-LTS L is a finite alternating sequence of states and actions, which starts with an initial state s, ends in some state s', and respects the transition relation $\to_f$. We denote by $traces(L)$ the set of all L's runs. Given a run r, $|r|$ denotes the number of states in r, $last(r)$ denotes r's last state, and r^i, where $1 \le i \le |r|$, denotes the run obtained by truncating r at the i-th state.

The relation $\to_f$ formalizes the system's small step operational semantics. Figure 3 shows three rules describing the successful execution of `SELECT` and `INSERT` commands, as well as triggers. In the rules, we represent context changes using the update function upd, which takes as input an M-state and an action $a \in \mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D$, and returns the updated context. This function, for instance, updates the system's history stored in the context. The function trg takes as input a system state s and returns the first trigger in the list of scheduled triggers stored in s's context. If there are no triggers to be executed, then $trg(s) = \epsilon$. The rule *SELECT Success* models the system's behaviour when the user u issues a `SELECT` query q that is authorized by the PDP f. The only component of the M-state s that changes is the context c. Namely, c' is obtained from c by updating the history and storing q's result. Similarly, the rule *INSERT Success* describes how the system behaves after a successful `INSERT` command, i.e., one that neither violates the integrity constraints nor causes security exceptions. The database state db is updated by adding the tuple $\overline{t}$ to R and the context is updated from c to c' by (a) storing the action's result, (b) storing the triggers that must be executed in response to the `INSERT` event, and (c) keeping track of the previous state in case a roll-back is needed.

database state s and a query $q := \{\overline{x} \mid \phi\}$, if the access control mechanism authorizes the boolean query $\bigwedge_{\overline{t}\in[q]^s} \phi[\overline{x} \mapsto \overline{t}] \wedge (\forall \overline{x}.\, \phi \Rightarrow \bigvee_{\overline{t}\in[q]^s} \overline{x} = \overline{t})$, then we return q's result, and otherwise we reject q as unauthorized.

The *Trigger INSERT Success* rule describes how the system executes a trigger whose action is an `INSERT`. The system extracts from the context the trigger t to be executed, i.e., $t = trg(s)$. It determines, using the function $user$, the user u under whose privileges the trigger t is executed, which is, depending on t's security mode, either the invoker $invoker(s)$ or t's owner. It then checks that u is authorized to execute the `SELECT` statement associated with t's `WHEN` condition, and that this condition is satisfied. Afterwards, it computes the actual action using the function act, which instantiates the free variables in t's definition with the values in the tuple $tpl(s)$, i.e., the tuple associated with the action that fired t. Finally, the system updates the database state db by adding the tuple $\overline{v}'$ to R and the context by storing the results of t's execution and removing t from the list of scheduled triggers.

In Appendix A, we give the complete formalization of our labelled transition system. This includes formalizing contexts and all the rules defining the transition relation $\to_f$. Our operational semantics can be tailored to model the behaviour of specific DBMSs. Thus, using our executable model, available at [26], it is possible to validate our operational semantics against different existing DBMSs.

4.3 Attacker Model

We model attackers that interact with the system through SQL commands and infer information from the system's behaviour by exploiting triggers, views, and integrity constraints. We argue that database access control mechanisms should be secure with respect to such strong attackers, as this reflects how (malicious) users may interact with modern databases. Furthermore, any mechanism secure against such strong attackers is also secure against weaker attackers.

Any user other than the administrator can be an attacker, and we assume that users do not collude to subvert the system. Note that our attacker model, the security properties in §5, and the mechanism we develop in §6, can easily be extended to support colluding users. We also assume that an attacker can issue any command available to the system's users, and he knows the system's operational semantics, the database schema, and the integrity constraints.

We assume that an attacker has access to the system's security policy, the set of users, and the definitions of the triggers and views in the system's state. In more detail, given an M-state $\langle db, U, sec, T, V, c\rangle$, an attacker can access U, sec, T, and V. Users interacting with existing DBMSs typically have access to some, although not all, of this information. For instance, in PostgreSQL a user can read all the information about the triggers defined on the tables for which he has some non-`SELECT` privileges. Note that the more information an attacker has, the more attacks he can launch. Finally, we assume that an attacker knows whether any two of his commands c and c' have been executed consec-

$$\frac{\begin{array}{c} s = \langle db, sec, U, T, V, c\rangle \quad f(s, \langle u, \mathtt{SELECT}, q\rangle) = \top \quad trg(s) = \epsilon \\ s' = \langle db, sec, U, T, V, c'\rangle \qquad c' = upd(s, \langle u, \mathtt{SELECT}, q\rangle) \end{array}}{s \xrightarrow{\langle u, \mathtt{SELECT}, q\rangle}_f s'} \ \begin{array}{c}\mathtt{SELECT} \\ \text{Success}\end{array}$$

$$\frac{\begin{array}{c} s = \langle db, sec, U, T, V, c\rangle \quad f(s, \langle u, \mathtt{INSERT}, R, \overline{t}\rangle) = \top \\ s' = \langle db[R \oplus \overline{t}], sec, U, T, V, c'\rangle \qquad db[R \oplus \overline{t}] \in \Omega_D^\Gamma \\ c' = upd(s, \langle u, \mathtt{INSERT}, R, \overline{t}\rangle) \qquad trg(s) = \epsilon \end{array}}{s \xrightarrow{\langle u, \mathtt{INSERT}, R, \overline{t}\rangle}_f s'} \ \begin{array}{c}\mathtt{INSERT} \\ \text{Success}\end{array}$$

$$\frac{\begin{array}{c} s = \langle db, sec, U, T, V, c\rangle \qquad \overline{v} = tpl(s) \\ u = user(m, owner, invoker(s)) \\ trg(s) = \langle id, owner, ev, R', \phi, st, m\rangle \\ f(s, \langle u, \mathtt{SELECT}, \phi[\overline{x} \mapsto \overline{v}]\rangle) = \top \quad [\phi[\overline{x} \mapsto \overline{v}]]^{db} = \top \\ \langle u, \mathtt{INSERT}, R, \overline{v}'\rangle = act(st, u, \overline{v}) \\ f(s, \langle u, \mathtt{INSERT}, R, \overline{v}'\rangle) = \top \qquad c' = upd(s, trg(s)) \\ s' = \langle db[R \oplus \overline{v}'], sec, U, T, V, c'\rangle \quad db[R \oplus \overline{v}'] \in \Omega_D^\Gamma \end{array}}{s \xrightarrow{trg(s)}_f s'} \ \begin{array}{c}\text{Trigger} \\ \mathtt{INSERT} \\ \text{Success}\end{array}$$

Figure 3: Examples of system model's rules.

$$\frac{\begin{array}{c} r^i = r^{i-1} \cdot \langle u, \mathtt{DELETE}, R, \overline{t}\rangle \cdot s \quad 1 < i \leq |r| \\ s \in \Omega_M \qquad secEx(s) = \perp \qquad Ex(s) = \emptyset \end{array}}{r, i \vdash_u \neg R(\overline{t})} \ \begin{array}{c}\mathtt{DELETE} \\ \text{Success}\end{array}$$

$$\frac{\begin{array}{c} r^i = r^{i-1} \cdot \langle u, \mathtt{SELECT}, \phi\rangle \cdot s \quad 1 < i \leq |r| \quad s \in \Omega_M \\ secEx(s) = \perp \qquad Ex(s) = \emptyset \qquad res(s) = \top \end{array}}{r, i \vdash_u \phi} \ \begin{array}{c}\mathtt{SELECT} \\ \text{Success}\end{array}$$

$$\frac{\begin{array}{c} r^{i+1} = r^i \cdot t \cdot s \quad invoker(last(r^i)) = u \quad s \in \Omega_M \quad 1 \leq i < |r| \\ secEx(s) = \perp \qquad Ex(s) = \emptyset \qquad r, i \vdash_u \neg\psi \qquad r, i+1 \vdash_u \psi \\ t = \langle id, ow, ev, R', \phi(\overline{x}), \langle \mathtt{INSERT}, R, \overline{t}\rangle, m\rangle \end{array}}{r, i \vdash_u \phi[\overline{x} \mapsto tpl(last(r^i))]} \ \begin{array}{c}\text{Learn} \\ \mathtt{INSERT} \\ \text{Backward}\end{array}$$

$$\frac{\begin{array}{c} r^{i+1} = r^i \cdot \langle u, \mathtt{SELECT}, \phi\rangle \cdot s \\ r, i+1 \vdash_u \psi \quad s \in \Omega_M \quad 1 \leq i < |r| \end{array}}{r, i \vdash_u \psi} \ \begin{array}{c}\text{Propagate} \\ \text{Backward} \\ \mathtt{SELECT}\end{array}$$

$$\frac{\begin{array}{c} r, i-1 \vdash_u \phi \qquad r^i = r^{i-1} \cdot \langle u, op, R, \overline{t}\rangle \cdot s \qquad s \in \Omega_M \qquad 1 < i \leq |r| \\ secEx(s) = \perp \quad Ex(s) = \emptyset \quad revise(r^{i-1}, \phi, r^i) = \top \quad op \in \{\mathtt{INSERT}, \mathtt{DELETE}\} \end{array}}{r, i \vdash_u \phi} \ \begin{array}{c}\text{Propagate Forward} \\ \text{Update Success}\end{array}$$

Figure 4: Example of attacker inference rules, where $r, i \vdash_u \phi$ denotes that this judgment holds in $\mathcal{ATK}_u$.

utively by the system, i.e., if there are commands executed by other users occurring between c and c'. The attacker's knowledge about the sequential execution of his commands is needed to soundly propagate his knowledge about the system's state between his commands. Since the mechanism we develop in §6 is secure with respect to this attacker, it is also secure with respect to weaker attackers who have less information or cannot detect whether their commands have been executed consecutively.

An attacker model describes what information an attacker knows, how he interacts with the system, and what he learns about the system's data by observing the system's behaviour. Since every user is a potential attacker, for each user $u \in \mathcal{U}$ we define an attacker model specifying u's inference capabilities. To represent u's knowledge, we introduce judgments. A judgment is a four-tuple $\langle r, i, u, \phi\rangle$, written $r, i \vdash_u \phi$, denoting that from the run r, which represents the system's behaviour, the user u can infer that ϕ holds in the i-th state of r. An attacker model for u is thus a set of judgments associating to each position of each run, the sentences that u can infer from the system's behaviour. The idea of representing the attacker's knowledge using sentences ϕ is inspired by existing formalisms for Inference Control [12, 20] and Controlled Query Evaluation [11].

Definition 4.2. Let P be an extended configuration, L be the P-LTS, and $u \in \mathcal{U}$ be a user. A (P, u)*-judgment* is a tuple $\langle r, i, u, \phi\rangle$, written $r, i \vdash_u \phi$, where $r \in traces(L)$, $1 \leq i \leq |r|$, and $\phi \in RC_{bool}$. A (P, u)*-attacker model* is a set of (P, u)-judgments. A (P, u)-judgment $r, i \vdash_u \phi$ *holds in a* (P, u)*-attacker model* A iff $r, i \vdash_u \phi \in A$. □

For each user $u \in \mathcal{U}$, we now define the (P, u)-attacker model $\mathcal{ATK}_u$ that we use in the rest of the paper. We formalize this model using a set of inference rules, where $\mathcal{ATK}_u$ is the smallest set of judgments satisfying the inference rules. Figure 4 shows five representative rules. The complete formalization of all rules is given in Appendix B.

In the following, when we say that a judgment $r, i \vdash_u \phi$ holds, we always mean with respect to the attacker model $\mathcal{ATK}_u$.

Note that $\mathcal{ATK}_u$ is sound with respect to the RC semantics, i.e., if $r, i \vdash_u \phi$ holds, then the formula ϕ holds in the i-th state of r. Intuitively, $\mathcal{ATK}_u$ models how u infers information from the system's behaviour, namely (a) how u learns information from his commands and their results, (b) how u learns information from triggers, their execution, their interleavings, and their side effects, (c) how u propagates his knowledge along a run, and (d) how u learns information from exceptions caused by either integrity constraint violations or security violations. This model is substantially more powerful than the `SELECT`-only attacker model.

The rules *`DELETE` Success* and *`SELECT` Success* describe how the user u infers information from his successful actions, i.e., those actions that generate neither security exceptions nor integrity violations. In the rules, $secEx(s) = \perp$ denotes that there were no security exceptions caused by the action leading to s, and $Ex(s) = \emptyset$ denotes that the action leading to s has not violated the integrity constraints. After a successful `DELETE`, u knows that the deleted tuple is no longer in the database, and after a successful `SELECT` he learns the query's result, denoted by $res(s)$.

The rules *Propagate Backward `SELECT`* and *Propagate Forward Update Success* describe how u propagates information along the run. *Propagate Backward `SELECT`* states that if the user u knows that ϕ holds after a `SELECT` command, then he knows that ϕ also holds just before the `SELECT` command because `SELECT` commands do not modify the database state. *Propagate Forward Update Success* states that if u knows that ϕ holds before a successful `INSERT` or `DELETE` command and he can determine that the command's execution does not influence ϕ's truth value, denoted by $revise(r^{i-1}, \phi, r^i) = \top$, then he also knows that ϕ holds after the command. The function *revise* is formalized in Appendix B.

Finally, the rule *Learn* ***INSERT*** *Backward* models u's reasoning when he activates a trigger that successfully inserts a tuple in the database. If u knows that immediately before the trigger the formula ψ does not hold and immediately after the trigger the formula ψ holds, then the trigger's execution is the cause of the database state's change. Therefore, u can infer that the trigger's condition ϕ holds just before the trigger's execution. Note that $invoker(s)$ denotes the user who fired the trigger that is executed in the state s, whereas $tpl(s)$ denotes the tuple associated with the action that fired the trigger that is executed in the state s.

Example 4.1. Let the schema, the set of users U, and the policy S be as in Example 3.1. The database state db is $db(N) = \{v\}$, $db(P) = \emptyset$, and $db(T) = \{v\}$. The only trigger in the system is $t = \langle id, admin, INS, P, T(x_1), \langle \texttt{INSERT}, N, x_1\rangle, O\rangle$. The run r is as follows:

1. u deletes v from N.
2. u inserts v in P. This activates the trigger t, which inserts v in N.
3. u issues the `SELECT` query $N(v)$.

We used Maude to generate the following run, which illustrates how the system's state changes. Note that there are no exceptions during the run.

$$\begin{tikzcd}
\langle db, U, S, \{t\}, \emptyset, c_1\rangle \arrow[r, "{\langle u, \texttt{DELETE}, N, v\rangle}"] & \langle db[N \ominus v], U, S, \{t\}, \emptyset, c_2\rangle \arrow[d, "{\langle u, \texttt{INSERT}, P, v\rangle}"'] \\
\langle db[P \oplus v], U, S, \{t\}, \emptyset, c_4\rangle \arrow[d, "{\langle u, \texttt{SELECT}, N(v)\rangle}"] & \langle db[P \oplus v, N \ominus v], U, S, \{t\}, \emptyset, c_3\rangle \arrow[l, "t"'] \\
\langle db[P \oplus v], U, S, \{t\}, \emptyset, c_5\rangle &
\end{tikzcd}$$

Figure 5 models u's reasoning in Attack 5. The user u first applies the ***SELECT*** *Success* rule to derive $r, 5 \vdash_u N(v)$, i.e., he learns the query's result. By applying the rule *Propagate Backward* ***SELECT*** to $r, 5 \vdash_u N(v)$, he obtains $r, 4 \vdash_u N(v)$, i.e., he learns that $N(v)$ holds before the `SELECT` query. Similarly, he applies the rule ***DELETE*** *Success* to derive $r, 2 \vdash_u \neg N(v)$, and he obtains $r, 3 \vdash_u \neg N(v)$ by applying the *Propagate Forward Update Success* rule. Finally, by applying the rule *Learn* ***INSERT*** *Backward* to $r, 3 \vdash_u \neg N(v)$ and $r, 4 \vdash_u N(v)$, he learns the value of the trigger's `WHEN` condition $r, 3 \vdash_u T(v)$. Since the user u should not be able to learn information about T, the attack violates the intended confidentiality guarantees. We used our executable attacker model [26] to derive the judgments. ■

5. SECURITY PROPERTIES

Here we define two security properties: database integrity and data confidentiality. These properties capture the two essential aspects of database security. Database integrity states that all actions modifying the system's state are authorized by the system's policy. In contrast, data confidentiality states that all information that an attacker can learn by observing the system's behaviour is authorized.

These two properties formalize security guarantees with respect to the two different classes of attacks previously identified. An access control mechanism providing database integrity prevents non-authorized changes to the system's state and, thereby, prevents integrity attacks. Similarly, by preventing the leakage of sensitive data, a mechanism providing data confidentiality prevents confidentiality attacks.

$$\frac{\begin{array}{c} s = \langle db, U, sec, T, V, c\rangle \quad u, o \in U \quad op \in \{\oplus, \oplus^*\} \\ priv = \langle \texttt{SELECT}, v\rangle \quad v = \langle id, o, q, O\rangle \quad v \in V \\ hasAccess(s, v, o, \oplus^*) \end{array}}{s \rightsquigarrow_{auth} \langle op, u, priv, o\rangle} \ \texttt{GRANT}$$

$$\frac{\begin{array}{c} s = \langle db, U, sec, T, V, c\rangle \quad t = \langle id, ow, ev, R, \phi, st, A\rangle \\ [\phi[\overline{x} \mapsto tpl(s)]]^{db} = \top \quad s \rightsquigarrow_{auth} act(st, ow, tpl(s)) \\ s \rightsquigarrow_{auth} act(st, invoker(s), tpl(s)) \qquad t \in T \end{array}}{s \rightsquigarrow_{auth} t} \ \texttt{TRIGGER}$$

$$\frac{\begin{array}{c} s = \langle db, U, sec, T, V, c\rangle \quad s' = \langle db, U, sec', T, V, c\rangle \\ s' = apply(\langle \ominus, u, p, u'\rangle, s) \quad \forall g \in sec'.\, s' \rightsquigarrow_{auth} g \end{array}}{s \rightsquigarrow_{auth} \langle \ominus, u, p, u'\rangle} \ \texttt{REVOKE}$$

Figure 6: Examples of $\rightsquigarrow_{auth}$ rules.

5.1 Database Integrity

Database integrity requires a formalization of authorized actions. We therefore define the relation $\rightsquigarrow_{auth}$ between states and actions, modelling which actions are authorized in a given state. Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ and f is an M-PDP. The relation $\rightsquigarrow_{auth} \subseteq \Omega_M \times (\mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D)$ is defined by a set of rules given in Appendix C. Figure 6 shows three representative rules. The ***GRANT*** rule says that the owner o of a view v with owner's privileges is authorized to delegate the `SELECT` privilege over v to a user u in the state s, if o has the `SELECT` privilege with grant option over a set of tables and views that determine v's materialization [34], denoted by $hasAccess(s, v, o, \oplus^*)$. The ***TRIGGER*** rule says that the execution of an enabled trigger, i.e., one whose `WHEN` condition is satisfied, with the activator's privileges is authorized if both the invoker and the trigger's owner are authorized to execute the trigger's action according to $\rightsquigarrow_{auth}$. Note that the *act* function instantiates the action given in the trigger's definition to a concrete action by identifying the user performing the action and replacing the free variables with values from **dom**. Finally, the ***REVOKE*** rule says that a `REVOKE` statement is authorized if the resulting state, obtained using the function *apply*, has a consistent policy, namely one in which all the `GRANT`s are authorized by $\rightsquigarrow_{auth}$.

We now define database integrity. Intuitively, a PDP provides database integrity iff all the actions it authorizes are explicitly authorized by the policy, i.e., they are authorized by $\rightsquigarrow_{auth}$. This notion comes directly from the SQL standard, and it is reflected in existing enforcement mechanisms. Recall that, given a state s, $secEx(s) = \bot$ denotes that there were no security exceptions caused by the action or trigger leading to s.

Definition 5.1. Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ and f is an M-PDP, and let L be the P-LTS. We say that f *provides database integrity with respect to* P iff for all reachable states $s, s' \in \Omega_M$, if s' is reachable in one step from s by an action $a \in \mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D$ and $secEx(s') = \bot$, then $s \rightsquigarrow_{auth} a$. □

Example 5.1. We consider a run corresponding to Attack 1, which illustrates a violation of database integrity. The database db is such that $db(P) = \emptyset$ and $db(S) = \{z\}$, the policy sec is $\{\langle \oplus, u_1, \langle \texttt{CREATE TRIGGER}, P\rangle, admin\rangle$, $\langle \oplus, u_2, \langle \texttt{INSERT}, P\rangle, admin\rangle$, $\langle \oplus, u_2, \langle \texttt{DELETE}, S\rangle, admin\rangle$, $\langle \oplus, u_2, \langle \texttt{SELECT}, P\rangle, admin\rangle$, $\langle \oplus, u_2, \langle \texttt{SELECT}, S\rangle, admin\rangle\}$, and the set U is $\{u_1, u_2, admin\}$. The run r is as follows:

$$\dfrac{\dfrac{\dfrac{}{r,2 \vdash_u \neg N(v)}\,\texttt{DELETE}\text{ Success}}{r,3 \vdash_u \neg N(v)}\,\text{Propagate Forward Update Success} \qquad \dfrac{\dfrac{}{r,5 \vdash_u N(v)}\,\texttt{SELECT}\text{ Success}}{r,4 \vdash_u N(v)}\,\text{Propagate Backward }\texttt{SELECT}}{r,3 \vdash_u T(v)}\,\text{Learn }\texttt{INSERT}\text{ Backward}$$

Figure 5: Template Derivation of Attack 5 (contains just selected subgoals)

1. The user u_1 creates the trigger $t = \langle id, u_1, INS, P, \top, \langle \texttt{DELETE}, S, z\rangle, A\rangle$.
2. The user u_2 inserts the value v in P. This activates the trigger t and deletes the content of S, i.e., the value z.

We used Maude to generate the following run, which illustrates how the system's state changes. Note that there are no exceptions during the run.

$$\begin{tikzcd}
\langle db, U, sec, \emptyset, \emptyset, c_1\rangle \arrow[r, "{\langle u_1, \texttt{CREATE}, t\rangle}"] & \langle db, U, sec, \{t\}, \emptyset, c_2\rangle \arrow[d, "{\langle u_2, \texttt{INSERT}, P, v\rangle}"'] \\
\langle db[P \oplus v, S \ominus z], U, sec, \{t\}, \emptyset, c_4\rangle & \langle db[P \oplus v], U, sec, \{t\}, \emptyset, c_3\rangle \arrow[l, "t"]
\end{tikzcd}$$

Access control mechanisms that do not restrict the execution of triggers with activator's privileges violate database integrity because they do not throw security exceptions when $\langle db[P \oplus v], U, sec, \{t\}, \emptyset, c_3\rangle \not\rightsquigarrow_{auth} t$. ■

5.2 Data Confidentiality

To model data confidentiality, we first introduce the concept of indistinguishability of runs, which formalizes the desired confidentiality guarantees by specifying whether users can distinguish between different runs based on their observations. Formally, a *P-indistinguishability relation* is an equivalence relation over $traces(L)$, where P is an extended configuration and L is the P-LTS. Indistinguishable runs, intuitively, should disclose the same information.

We now define the concept of a secure judgment, which is a judgment that does not leak sensitive information or, equivalently, one that cannot be used to differentiate between indistinguishable runs.

Definition 5.2. Let P be an extended configuration, L be the P-LTS, and $\cong$ be a P-indistinguishability relation. A judgment $r, i \vdash_u \phi$ is *secure with respect to P and $\cong$*, written $secure_{P,\cong}(r, i \vdash_u \phi)$, iff for all $r' \in traces(L)$ such that $r^i \cong r'$, it holds that $[\phi]^{db} = [\phi]^{db'}$, where $last(r^i) = \langle db, U, S, T, V, c\rangle$ and $last(r') = \langle db', U', S', T', V', c'\rangle$. □

We are now ready to define data confidentiality. Intuitively, an access control mechanism provides data confidentiality iff all judgments that an attacker can derive are secure.

Definition 5.3. Let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, $u \in \mathcal{U}$ be a user, A be a (P, u)-attacker model, and $\cong$ be a P-indistinguishability relation. We say that *f provides data confidentiality with respect to P, u, A, and* $\cong$ iff $secure_{P,\cong}(r, i \vdash_u \phi)$ for all judgments $r, i \vdash_u \phi$ that hold in A. □

We now define the indistinguishability relation that we use in the rest of the paper, which captures what each user can observe (as stated in §4.3) and the effects of the system's access control policy. Let $P = \langle\langle D, \Gamma\rangle, f\rangle$ be an extended configuration, L be the P-LTS, and u be a user in $\mathcal{U}$. Given a run $r \in traces(L)$, the user u is aware only of his actions and not of the actions of the other users in r. This is represented by the u-projection of r, which is obtained by masking all sequences of actions that are not issued by u using a distinguished symbol $*$. Specifically, the *u-projection of* r is a sequence of states in Ω_M and actions in $\mathcal{A}_{D,u} \cup \mathcal{TRIGGER}_D \cup \{*\}$ that is obtained from r by (1) replacing each action not issued by u with $*$, (2) replacing each trigger whose invoker is not u with $*$, and (3) replacing all non-empty sequences of $*$-transitions with a single $*$-transition. For each user $u \in \mathcal{U}$, we define the P-indistinguishability relation $\cong_{P,u}$, which is formally defined in Appendix D. Intuitively, two runs r and r' are $\cong_{P,u}$-indistinguishable, denoted $r \cong_{P,u} r'$, iff (1) the labels of the u-projections of r and r' are the same, (2) u executes the same actions $a_1, \ldots, a_n$ in r and r', in the same order, and with the same results, and (3) before each action a_i, where $1 \leq i \leq n$, as well as in the last states of r and r', the views, the triggers, the users, and the data disclosed by the policy are the same in r and r'.

We remark that there is a close relation between $\cong_{P,u}$ and state-based indistinguishability [24,35,46]. For any two $\cong_{P,u}$-indistinguishable runs r and r', the database states that precede all actions issued by u as well as the last states in r and r' are pairwise indistinguishable under existing state-based notions [24,35,46].

Example 5.2 illustrates our indistinguishability notion.

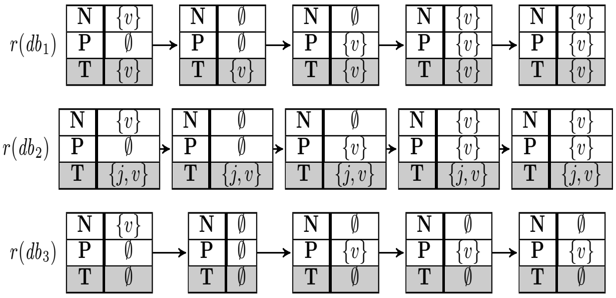


Figure 7: The runs $r(db_1)$ and $r(db_2)$ are indistinguishable, whereas $r(db_1)$ and $r(db_3)$ are not.

Example 5.2. Let the schema, the set of users, the policy, and the triggers be as in Example 4.1. Consider the following run $r(db)$, parametrized by the initial database state db:

1. u deletes v from N.
2. u inserts v in P. If v is in T, this activates the trigger t, which, in turn, inserts v in N.
3. u issues the `SELECT` query $N(v)$.

Let db_1, db_2, and db_3 be three database states such that $db_1(T) = \{v\}$, $db_2(T) = \{j, v\}$, and $db_3(T) = \emptyset$, whereas $db_i(N) = \{v\}$ and $db_i(P) = \emptyset$, for $1 \leq i \leq 3$. Note that $r(db_1)$ is the run used in Example 4.1. Figure 7 depicts how the database's state changes during the runs $r(db_i)$, for $1 \leq i \leq 3$. Gray indicates those tables that the user u cannot read. The runs $r(db_1)$ and $r(db_2)$ are indistinguishable for the user u. The only difference between them is the content of the table T, which u cannot read. In contrast, u can distinguish between $r(db_1)$ and $r(db_3)$ because the trigger has been executed in the former and not in the latter.

Indistinguishability may also depend on the actions of the other users. Consider the runs r' and r'' obtained by extending $r(db_1)$ respectively with one and two `SELECT` queries issued by the administrator just after u's query. The user u can distinguish between $r(db_1)$ and r' because he knows that other users interacted with the system in r' but not in $r(db_1)$, i.e., the u-projections have different labels. In contrast, the runs r' and r'' are indistinguishable for u because he only knows that, after his own `SELECT`, other users interacted with the system, i.e., the u-projections have the same labels. However, he does not know the number of commands, the commands themselves, or their results. ■

Example 5.3 shows that existing PDPs leak sensitive information and therefore do not provide data confidentiality.

Example 5.3. In Example 4.1, we showed how the user u derives $r, 3 \vdash_u T(v)$. The judgment is not secure because there is a run indistinguishable from r^3, i.e., the run $r^3(db_3)$ in Example 5.2, in which $T(v)$ does not hold. ■

Example 5.4 shows how views may leak information about the underlying tables. Even though this leakage might be considered legitimate, there is no way in our setting to distinguish between intended and unintended leakages. If this is desired, data confidentiality can be extended with the concept of *declassification* [6, 7].

Example 5.4. Consider a database with two tables T and Z and a view $V = \langle v, admin, \{x \mid T(x) \wedge Z(x)\}, O\rangle$. The set U is $\{u, admin\}$ and the policy S is $\{\langle \oplus, u, \langle \texttt{SELECT}, T\rangle, admin\rangle, \langle \oplus, u, \langle \texttt{SELECT}, V\rangle, admin\rangle, \langle \oplus, u, \langle \texttt{INSERT}, T\rangle, admin\rangle\}$. Consider the following run r, parametrized by the initial database state db, where u first inserts 27 into T and afterwards issues the `SELECT` query $V(27)$. We assume there are no exceptions in r.

$$\begin{tikzcd}
\langle db, U, S, \emptyset, \{V\}, c_1\rangle \arrow[r, "{\langle u, \texttt{INSERT}, T, 27\rangle}"] & \langle db[T \oplus 27], U, S, \emptyset, \{V\}, c_2\rangle \arrow[d, "{\langle u, \texttt{SELECT}, V(27)\rangle}"'] \\
& \langle db[T \oplus 27], U, S, \emptyset, \{V\}, c_3\rangle
\end{tikzcd}$$

We used Maude to generate the runs $r(d)$ and $r(d')$ with the initial database states d and d' such that $d(T) = d(Z) = d'(T) = \emptyset$ and $d'(Z) = \{27\}$. The runs $r^1(d)$ and $r^1(d')$ are indistinguishable for u because they differ only in the content of Z, which u cannot read. After the `INSERT`, u can distinguish between $r^2(d)$ and $r^2(d')$ by reading V. Indeed, $d[T \oplus 27](V) = \emptyset$, because $d(Z) = \emptyset$, whereas $d'[T \oplus 27](V) = \{27\}$. The user u derives $r(d'), 1 \vdash_u Z(27)$, which is not secure because $r^1(d)$ and $r^1(d')$ are indistinguishable for u, but $Z(27)$ holds just in the latter. ■

In contrast to existing security notions [24,35,46], we have defined data confidentiality over runs. This is essential to model and detect attacks, such as those in Examples 5.3 and 5.4, where an attacker infers sensitive information from the transitions between states. For instance, the leakage in Example 5.4 is due to the execution of the `INSERT` command. Although the `SELECT` command is authorized by the policy, u can use it to infer sensitive information about the system's state before the `INSERT` execution.

6. A PROVABLY SECURE PDP

We now present a PDP that provides both database integrity and data confidentiality. We first explain the ideas behind it using examples. Afterwards, we show that it satisfies the desired security properties and has acceptable overhead. Finally, we argue that it is more permissive than existing access control solutions.

Figure 8 depicts our PDP f together with the functions f_{int} and f_{conf}. Additional details about the PDP are given in Appendices F–G. The PDP takes as input a state s and an action a and outputs $\top$ iff both f_{int} and f_{conf} authorize a in s, i.e., iff a's execution neither violates database integrity nor data confidentiality. Note that our algorithm is not *complete* in that it may reject some secure commands. However, from the results in [24, 30, 34], it follows that no algorithm can be complete and provide database integrity and data confidentiality for the relational calculus.

Our PDP is invoked by the database system each time a user u issues an action a to check whether u is authorized to execute a. The PDP is also invoked whenever the database system executes a scheduled trigger t: once to check if the `SELECT` statement associated with t's `WHEN` condition is authorized and once, in case t is enabled, to check if t's action is authorized.

6.1 Enforcing Database Integrity

The function f_{int} takes as input a state s and an action a. If the system is not executing a trigger, denoted by $trg(s) = \epsilon$, f_{int} checks (line 1) whether a is authorized with respect to s. In line 2, f_{int} checks whether a is the current trigger's condition. If this is the case, it returns $\top$ because the triggers' conditions do not violate database integrity. Finally, the algorithm checks (line 3) whether a is the current trigger's action, and if this is the case, it checks whether the current trigger $trg(s)$ is authorized with respect to s. The function *auth*, which checks if a is authorized with respect to s, is a sound and computable under-approximation of $\rightsquigarrow_{auth}$. Thus, any action authorized by f_{int} is authorized according to $\rightsquigarrow_{auth}$. This ensures database integrity. Note that $\rightsquigarrow_{auth}$ relies on the concept of *determinacy* [34] to decide whether a query is determined by a set of views. Since determinacy is undecidable [34], in *auth* we implement a sound under-approximation of it, given in Appendix E, that checks syntactically if a query is determined by a set of views.

Example 6.1. Consider a database with three tables: R, T, and Z. The set U is $\{u, u', admin\}$ and the policy S is $\{\langle \oplus, u, \langle \texttt{SELECT}, R\rangle, admin\rangle, \langle \oplus^*, u, \langle \texttt{SELECT}, T\rangle, admin\rangle, \langle \oplus^*, u, \langle \texttt{SELECT}, Z\rangle, admin\rangle\}$. There are two views $V = \langle v, admin, \{x \mid T(x) \wedge Z(x)\}, O\rangle$ and $W = \langle w, u, \{x \mid R(x) \vee V(x)\}, O\rangle$. The user u tries to grant to u' read access to W, i.e., he issues $\langle \oplus, u', \langle \texttt{SELECT}, W\rangle, u\rangle$. The PDP f_{int} rejects the command and raises a security exception because u is authorized to delegate the read access only for T and Z but W's result depends also on R, for which u cannot delegate read access. Assume now that the policy is $\{\langle \oplus^*, u, \langle \texttt{SELECT}, R\rangle, admin\rangle, \langle \oplus^*, u, \langle \texttt{SELECT}, T\rangle, admin\rangle, \langle \oplus^*, u, \langle \texttt{SELECT}, Z\rangle, admin\rangle\}$. In this case, f_{int} authorizes the `GRANT`. The reason is that W's definition can be equivalently rewritten as $\{x \mid R(x) \vee (T(x) \wedge Z(x))\}$ and u is authorized to delegate the read access for R, T, and Z. ■

$\triangleright$ *s is a state and a is an action*

function $f(s, a)$

return $f_{int}(s, a) \wedge f_{conf}(s, a, user(s, a))$

$\triangleright$ *s is a state and a is an action*

function $f_{int}(s, a)$

if $trg(s) = \epsilon$ **return** $auth(s, a)$

  else if $a = cond(trg(s), s)$ **return** $\top$

    else if $a = act(trg(s), s)$ **return** $auth(s, trg(s))$

      else return $\perp$

$\triangleright$ *s is a state, a is an action, and u is a user*

function $f_{conf}(s, a, u)$

1. **switch** a
2. **case** $\langle u', \texttt{SELECT}, q\rangle$: **return** $secure(u, q, s)$
3. **case** $\langle u', \texttt{INSERT}, R, \bar{t}\rangle$: **case** $\langle u', \texttt{DELETE}, R, \bar{t}\rangle$:
4. **if** $leak(a, s, u) \vee \neg secure(u, getInfo(a), s)$ **return** $\perp$
5. **for** $\gamma \in Dep(a, \Gamma)$
6. **if** $(\neg secure(u, getInfoS(\gamma, a), s) \vee \neg secure(u, getInfoV(\gamma, a), s))$
7. **return** $\perp$
8. **case** $\langle \oplus, u'', pr, u'\rangle, \langle \oplus^*, u'', pr, u'\rangle$: **return** $\neg leak(a, s, u)$
9. **return** $\top$

Figure 8: The PDP f uses the two subroutines f_{int} and f_{conf}. The former provides database integrity and the latter provides data confidentiality with respect to the user $user(s, a)$, which denotes either the user issuing the action, when the system is not executing a trigger, or the trigger's invoker.

6.2 Enforcing Data Confidentiality

The function f_{conf}, shown in Figure 8, takes as input an action a, a state s, and a user u. Note that any user other than the administrator is a potential attacker. The requirement for f_{conf} is that it authorizes only those commands that result in secure judgments for u as required by Definition 5.3. To achieve this, f_{conf} over-approximates the set of judgments that u can derive from a's execution. For instance, the algorithm assumes that u can always derive the trigger's condition from the run, even though this is not always the case. Then, f_{conf} authorizes a iff it can determine that all u's judgements are secure. This can be done by analysing just a finite subset of the over-approximated set of u's judgments.

In more detail, f_{conf} performs a case distinction on the action a (line 1). If a is a `SELECT` command (line 2), f_{conf} checks whether the query is secure with respect to the current state s and the user u using the *secure* procedure. If a is an `INSERT` or `DELETE` command (lines 3–7), f_{conf} checks (line 4), using the *leak* procedure, whether a's execution may leak sensitive information through the views that u can read, as in Example 5.4. Afterwards, f_{conf} also checks (line 4) whether the information u can learn from a's execution, modelled by the sentence computed by the procedure *getInfo*(a), is secure. In line 5–7, f_{conf} computes the set of all integrity constraints that a's execution may violate, denoted by $Dep(a, \Gamma)$, and for all constraints γ, it checks whether the information that u may learn from γ is secure. The procedure *getInfoS* (respectively *getInfoV*) computes the sentence modelling the information learned by u from γ if a is executed successfully (respectively violates γ). If a is a `GRANT` command (line 8), f_{conf} checks whether a's successful execution discloses sensitive information to u. In the remaining cases (line 9), f_{conf} authorizes a.

Secure judgments. Determining if a given judgment is secure is undecidable for RC [24, 30]. Hence, the *secure* procedure implements a sound and computable under-approximation of this notion. We now present our solution. Other sound under-approximations can alternatively be used without affecting f_{conf}'s data confidentiality guarantees.

Let $M = \langle D, \Gamma\rangle$ be a system configuration, $r, i \vdash_u \phi$ be a judgment, and $s = \langle db, U, sec, T, V, c\rangle$ be the i-th state in r. As a first under-approximation, instead of the set of all runs indistinguishable from r^i, we consider the larger set of all runs r' whose last state $s' = \langle db', U, sec, T, V, c'\rangle$ is such that the disclosed data in db and db' are the same. Note that if a judgment is secure with respect to this larger set, it is secure also with respect to the set of indistinguishable runs because the former set contains the latter. This larger set depends just on the database state db and the policy sec, not on the run or the attacker model $\mathcal{ATK}_u$. Determining judgment's security is, however, still undecidable even on this larger set. We therefore employ a second under-approximation that uses query rewriting. We rewrite the sentence ϕ to a sentence ϕ_{rw} such that if $r, i \vdash_u \phi$ is not secure for the user u, then $[\phi_{rw}]^{db} = \top$. The formula ϕ_{rw} is $\neg\phi^{\top}_{s,u} \wedge \phi^{\perp}_{s,u}$, where $\phi^{\top}_{s,u}$ and $\phi^{\perp}_{s,u}$ are defined inductively over ϕ. A formal definition of *secure* is given in Appendix F.

We now explain how we construct $\phi^{\top}_{s,u}$ and $\phi^{\perp}_{s,u}$. We assume that both ϕ and V contain only views with the owner's privileges. The extension to the general case is given in Appendix F. First, for each table or view $o \in D \cup V$, we create additional views representing any possible projection of o. The *extended vocabulary* contains the tables in D, the views in V, and their projections. For instance, given a table $R(x, y)$, we create the views R_x and R_y representing respectively $\{y \mid \exists x. R(x, y)\}$ and $\{x \mid \exists y. R(x, y)\}$. Second, we compute the formula ϕ' by replacing each sub-formula $\exists \overline{x}. R(\overline{x}, \overline{y})$ in ϕ with the view $R_{\overline{x}}(\overline{y})$ associated with the corresponding projection. Third, for each predicate R in the formula ϕ', we compute the sets $R^{\top}_{s,u}$ and $R^{\perp}_{s,u}$. The set $R^{\top}_{s,u}$ (respectively $R^{\perp}_{s,u}$) contains all the tables and views K in the extended vocabulary such that (1) K is contained in (respectively contains) R, and (2) the user u is authorized to read K in s, i.e., there is a grant $\langle op, u, \langle \texttt{SELECT}, K'\rangle, u'\rangle \in sec$ such that either $K' = K$ or K is obtained from K' through a projection. The formula $\phi^{v}_{s,u}$, where $v \in \{\top, \perp\}$, is:

$$\phi^{v}_{s,u} = \begin{cases} \bigvee_{S \in R^{\top}_{s,u}} S(\overline{x}) & \text{if } \phi = R(\overline{x}) \text{ and } v = \top \\ \bigwedge_{S \in R^{\perp}_{s,u}} S(\overline{x}) & \text{if } \phi = R(\overline{x}) \text{ and } v = \perp \\ \neg\psi^{\neg v}_{s,u} & \text{if } \phi = \neg\psi \\ \psi^{v}_{s,u} * \gamma^{v}_{s,u} & \text{if } \phi = \psi * \gamma \text{ and } * \in \{\vee, \wedge\} \\ Q\,x.\,\psi^{v}_{s,u} & \text{if } \phi = Q\,x.\,\psi \text{ and } Q \in \{\exists, \forall\} \\ \phi & \text{otherwise} \end{cases}$$

The formulae are such that if $\phi^{\top}_{s,u}$ holds, then ϕ holds and if $\neg\phi^{\perp}_{s,u}$ holds, then $\neg\phi$ holds. To compute the sets $R^{\top}_{s,u}$ and $R^{\perp}_{s,u}$, we check the containment between queries. Since query containment is undecidable [3], we implement a sound

Database State

S	
1	1
2	3
4	2

R
3

Q
4

Views

V	
1	1
2	3

W
3
4

$V = \{x, y \mid S(x, y) \wedge (x = 1 \vee y = 3)\}$

$W = \{x \mid R(x) \vee Q(x)\}$

Extended Vocabulary

$\mathbf{S}_x$
1
3
2

$\mathbf{S}_y$
1
2
4

$\mathbf{V}_x$
1
3

$\mathbf{V}_y$
1
2

Containment Sets

${S_y}^{\top}_{s,u} = \{V_y\}$ $R^{\top}_{s,u} = \emptyset$

${S_y}^{\perp}_{s,u} = \emptyset$ $R^{\perp}_{s,u} = \{W\}$

$S_x = \{y \mid \exists x.\, S(x, y)\}$ $V_x = \{y \mid \exists x.\, V(x, y)\}$

$S_y = \{x \mid \exists y.\, S(x, y)\}$ $V_y = \{x \mid \exists y.\, V(x, y)\}$

Original Sentence

$\phi := (\exists y.\, S(2, y)) \wedge (\neg R(5) \vee \exists y.\, S(4, y)) \equiv S_y(2) \wedge (\neg R(5) \vee S_y(4))$

Rewriting

$\phi_{rw} := \neg\phi^{\top}_{s,u} \wedge \phi^{\perp}_{s,u}$

$\phi^{\top}_{s,u} := S_y(2)^{\top}_{s,u} \wedge (\neg R(5)^{\perp}_{s,u} \vee S_y(4)^{\top}_{s,u}) \equiv V_y(2) \wedge (\neg W(5) \vee V_y(4))$

$\phi^{\perp}_{s,u} := S_y(2)^{\perp}_{s,u} \wedge (\neg R(5)^{\top}_{s,u} \vee S_y(4)^{\perp}_{s,u}) \equiv \top$

Figure 9: Checking the security of the judgment $r, 1 \vdash_u (\exists y.\, S(2, y)) \wedge (\neg R(5) \vee \exists y.\, S(4, y))$ from Example 6.2.

under-approximation of it, described in Appendix F. Other sound under-approximations can be used as well.

Our $\phi^{\top}_{s,u}$ and $\phi^{\perp}_{s,u}$ rewritings share similarities with the *low* and *high evaluations* of Wang et al. [46]. Both try to approximate the result of a query just by looking at the authorized data. However, we use $\phi^{\top}_{s,u}$ and $\phi^{\perp}_{s,u}$ to determine a judgment's security, whereas Wang et al. use evaluations to restrict the query's results only to authorized data.

Example 6.2. Consider a database with three tables S, R, and Q, and two views $V = \langle v, admin, \{x, y \mid S(x, y) \wedge (x = 1 \vee y = 3)\}, O\rangle$ and $W = \langle w, admin, \{x \mid R(x) \vee Q(x)\}, O\rangle$. The database state db is $db(S) = \{(1, 1), (2, 3), (4, 2)\}$, $db(R) = \{3\}$, and $db(Q) = \{4\}$, the set U is $\{u, admin\}$, and the policy sec is $\{\langle\oplus, u, \langle \texttt{SELECT}, V\rangle, admin\rangle, \langle\oplus, u, \langle\texttt{SELECT}, W\rangle, admin\rangle\}$. Let the state s be $\langle db, U, sec, \emptyset, \{V, W\}, \epsilon\rangle$ and the run r be s. We want to check the security of $r, 1 \vdash_u \phi$, where $\phi := (\exists y.\, S(2, y)) \wedge (\neg R(5) \vee \exists y.\, S(4, y))$, for the user u. Figure 9 depicts the database state db, the materializations of the views V and W, and the materializations of the views S_x, S_y, V_x, and V_y in the extended vocabulary. Gray indicates those tables and views that u cannot read.

The rewriting process, depicted also in Figure 9, proceeds as follows. We first rewrite the formula ϕ as $S_y(2) \wedge (\neg R(5) \vee S_y(4))$. The sets ${S_y}^{\top}_{s,u}$, ${S_y}^{\perp}_{s,u}$, $R^{\top}_{s,u}$, and $R^{\perp}_{s,u}$ are respectively $\{V_y\}$, $\emptyset$, $\emptyset$, and $\{W\}$. The formulae $\phi^{\top}_{s,u}$ and $\phi^{\perp}_{s,u}$ are respectively $S_y(2)^{\top}_{s,u} \wedge (\neg R(5)^{\perp}_{s,u} \vee S_y(4)^{\top}_{s,u})$, which is equivalent to $V_y(2) \wedge (\neg W(5) \vee V_y(4))$, and $S_y(2)^{\perp}_{s,u} \wedge (\neg R(5)^{\top}_{s,u} \vee S_y(4)^{\perp}_{s,u})$, which is equivalent to $\top$. They are both secure, as they depend only on V and W. Furthermore, since $\phi^{\top}_{s,u}$ holds in s, then ϕ holds as well. Finally, ϕ_{rw} is $\neg\phi^{\top}_{s,u} \wedge \phi^{\perp}_{s,u}$. Since ϕ_{rw} does not hold in s, it follows that $r, 1 \vdash_u \phi$ is secure. ■

6.3 Theoretical Evaluation

Our PDP provides the desired security guarantees and its data complexity, i.e., the complexity of executing f when the action, the policy, the triggers, and the views are fixed, is AC^0. This means that f can be evaluated in logarithmic space in the database's size, as $AC^0 \subseteq LOGSPACE$, and evaluation is highly parallelizable. Note that *secure*'s data complexity is AC^0 because it relies on query evaluation, whose data complexity is AC^0 [3]. In contrast, all other operations in f are executed in constant time in terms of data complexity. Note also that on a single processor, f's data complexity is polynomial in the database's size. We believe that this is acceptable because the database is usually very large, whereas the query, which determines the degree of the polynomial, is small. The proofs are given in Appendices F–G.

THEOREM 6.1. *Let $P = \langle M, f\rangle$ be an extended configuration, where M is a system configuration and f is as above. The PDP f (1) provides data confidentiality with respect to P, u, $\mathcal{ATK}_u$, and $\cong_{P,u}$, for any user $u \in \mathcal{U}$, and (2) provides database integrity with respect to P. Moreover, the data complexity of f is AC^0.*

As the Examples 6.3 and 6.4 below show, f is more permissive than existing PDPs for those actions that violate neither database integrity nor data confidentiality.

Example 6.3. Our PDP is more permissive than existing mechanisms for commands of the form `GRANT SELECT ON` V `TO` u, where V is a view with owner's privileges, u is a user, and the statement is issued by the view's owner o. Such mechanisms, in general, authorize the `GRANT` iff o is authorized to delegate the read permission for all tables and views that occur in v's definition. Consider again Example 6.1. Our PDP authorizes $\langle\oplus, u', \langle\texttt{SELECT}, W\rangle, u\rangle$ under the policy S'. However, existing mechanisms reject it because u is not directly authorized to read V, although u can read the underlying tables. Our PDP also authorizes all the secure `GRANT` statements authorized by existing mechanisms. ■

Example 6.4. Our PDP is more permissive than the mechanisms used in existing DBMSs for secure `SELECT` statements. Such mechanisms, in general, authorize a `SELECT` statement issued by a user u iff u is authorized to read all tables and views used in the query. They will reject the query in Example 6.2 even though the query is secure. Furthermore, any secure `SELECT` statement authorized by them will be authorized by our solution as well. Also the PDP proposed by Rizvi et al. [35] rejects the query in Example 6.2 as insecure. However, our solution and the proposal of Rizvi et al. [35] are incomparable in terms of permissiveness, i.e., some secure `SELECT` queries are authorized by one mechanism and not by the other. ■

6.4 Implementation

To evaluate the feasibility and security of our approach in practice, we implemented our PDP in Java. The prototype, available at [26], implements both our PDP and the operational semantics of our system model. It relies on the underlying PostgreSQL database for executing the `SELECT`, `INSERT`, and `DELETE` commands. Note that our prototype also handles all the differences between the relational calculus and SQL. For instance, it translates every relational calculus query into an equivalent `SELECT` SQL query over the underlying database. We performed a preliminary experimental evaluation of our prototype. Our experiments were run on a PC with an Intel i7 processor and 32GB of RAM. Note that we materialized the content of all the views.

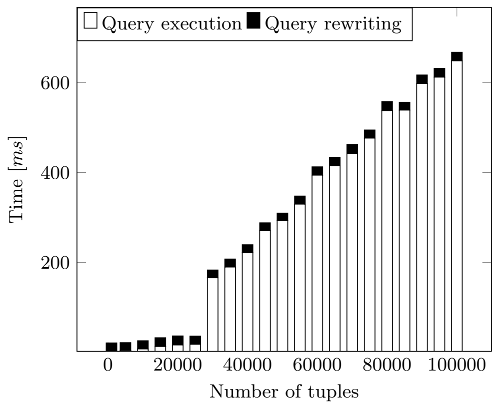
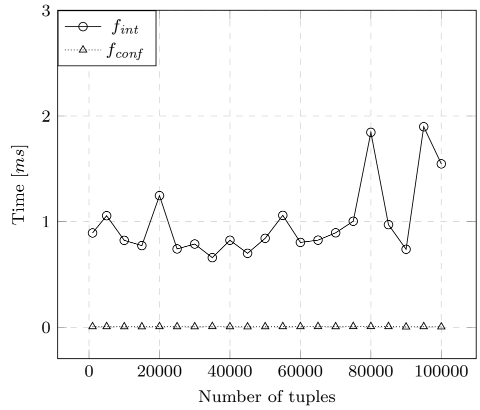
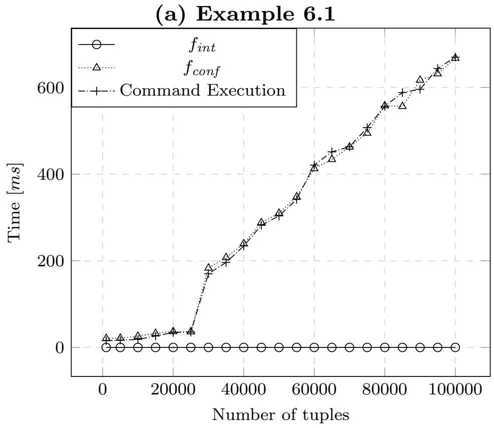


Figure 11: Example 8 : f_{conf}'s execution time.



(a) Example 6.1



(b) Example 6.2

Figure 10: PDP Execution time.

Our evaluation has two objectives: (1) to empirically validate that the prototype provides the desired security guarantees, and (2) to evaluate its overhead. For (1), we ran the attacks in §2 against our prototype. As expected, our PDP prevents all the attacks. For (2), we simulated Examples 6.1 and 6.2 on database states where the number of tuples ranges from 1,000 to 100,000. Figure 10 shows the PDP's execution time. Our results show that our solution is feasible. In more detail, f_{int}'s execution time does not depend on the database size, whereas f_{conf}'s execution time does. We believe that the overhead introduced by the PDP is acceptable for a proof of concept. Even with 100,000 tuples, the PDP's running time is under a second. In Example 6.2, f_{conf}'s execution time is comparable to the execution time of the user's query. As Figure 11 shows, f_{conf}'s query rewriting time does not depend on the database's size, whereas f_{conf}'s query execution time does.

To improve f_{conf}'s performance, one could strike a different balance between simple syntactic checks and our query rewriting solution. This, however, would result in more restrictive PDPs. We will investigate further optimizations as a future work.

7. RELATED WORK AND DISCUSSION

We compare our work against two lines of research: database access control and information flow control. Both of these have similar goals, namely preventing the leakage and corruption of sensitive information.

7.1 Database Access Control

Discretionary Database Access Control. Our framework builds on prior research in database access control [24,35,46] as well as established notions from database theory, such as determinacy [34] and instance-based determinacy [30].

Specifically, our notion of secure judgments extends instance based determinacy from database states to runs, while data confidentiality extends existing security notions [24,35,46] to dynamic settings, where both the database and the policy may change. Similarly, our indistinguishability notion extends those in [24,46] from database states to runs. Finally, our formalization of $\rightsquigarrow_{auth}$ relies on determinacy to decide whether the content of a view is fully determined by a set of other views.

Griffiths and Wade propose a PDP [23] that prevents Attacks 2 and 3 by using syntactic checks and by removing all views whose owners lack the necessary permissions. In contrast, we prevent the execution of `GRANT` and `REVOKE` operations leading to inconsistent policies.

Mandatory Database Access Control. Research on mandatory database access control has historically focused on Multi-Level Security (MLS) [17], where both the data and the users are associated with security levels, which are compared to control data access. Our PDP extends the SQL discretionary access control model with additional *mandatory* checks to provide database integrity and data confidentiality. In the following, we compare our work with the access control policies and semantics used by MLS systems.

With respect to policies, our work uses the SQL access control model, where policies are sets of `GRANT` statements. In this model, users can dynamically modify policies by delegating permissions. In contrast, MLS policies are usually expressed by labelling each subject and object in the system with labels from a security lattice [37]. The policy is, in general, fixed (cf. the *tranquillity principle* [37]).

With respect to semantics, existing MLS solutions are based on the so-called *Truman model* [35], where they transparently modify the commands issued by the users to restrict the access to only the authorized data. In contrast, we use the same semantics as SQL, that is, we execute only the secure commands. This is called the *Non-Truman model* [35]. For an in-depth comparison of these access control models, see [24, 35]. Operationally, MLS mechanisms use poly-instantiation [29], which is neither supported by the relational model nor by the SQL standard, and requires ad-hoc extensions [17, 38]. Furthermore, the operational semantics of MLS systems differs from the standard relational semantics. In contrast, our operational semantics supports the relational model and is directly inspired by SQL.

The above differences influence how security properties are expressed. Data confidentiality, which relies on a precise characterization of security based on a possible worlds semantics, is a key component of the Non-Truman model (and SQL) access control semantics. Similarly, database integrity requires that any "write" operation is authorized according to the policy and is directly inspired by the SQL access control semantics. The security properties in MLS systems, in contrast, combine the multilevel relational semantics [17, 38] with MLS and BIBA properties [37].

MLS systems prevent attacks similar to Attacks 4 and 5 using poly-instantiated tuples and triggers [38, 42], whereas attacks similar to Attack 1 cannot be carried out because triggers with activator's privileges are not supported [42]. The SeaView system [17], which combines discretionary access control and MLS, additionally prevents attacks similar to Attacks 2 and 3 by relying on Griffiths and Wade's PDP [23]. However, these solutions cannot be applied to SQL databases for the aforementioned reasons.

7.2 Information Flow Control

Various authors have applied ideas from information flow control to databases. Davis and Chen [16] study how cross-application information flows can be tracked through databases. Other researchers [15, 32, 39] present languages for developing secure applications that use databases. They employ secure type systems to track information flows through databases. However, they neither model nor prevent the attacks we identified because they do not account for the advanced database features and the strong attacker model we study in this paper.

Schultz and Liskov [40] extend decentralized information flow control [33] to databases, based on concepts from multi-level security [17]. They identify one attack on data confidentiality that exploits integrity constraints. Their solution relies on poly-instantiation [29] and cannot be applied to SQL databases that do not support multi-level security. Their mechanism neither prevents the other attacks we identify nor provides provable and precise security guarantees.

Several researchers have studied attacker models in information flow control [5, 21]. Giacobazzi and Mastroeni [21] model attackers as data-flow analysers that observe the program's behaviour, whereas Askarov and Chong [5] model attackers as automata that observe the program's events. They both model passive attackers, who can observe, but do not influence, the program's execution. In contrast, our attacker is active and interacts with the database.

7.3 Discussion

Historically, database access control and information flow control rely on different foundations, formalisms, security notions, and techniques. We see our paper as a starting point for bridging these topics: we combine database access control theory with an operational semantics and an attacker model, which are common in information flow control, but have been largely ignored in database access control. We thereby give a precise logical characterization of the attacker's capabilities and of a judgment's security. Furthermore, our indistinguishability notion has similarities with the low-equivalence notions used in [6, 7, 10], whereas both data confidentiality and the notion of secure judgments have a precise characterization as instances of non-interference [22, 36]; see Appendix H for more details.

We believe our framework provides a basis for (1) further investigating the connections between these two topics, (2) applying information flow mechanisms, such as type systems or multi-execution [18], to database access control, and (3) investigating how integrity notions used in information flow control can best be applied to databases.

8. CONCLUSION

Motivated by practical attacks against existing databases, we have initiated several new research directions. First, we developed the idea that attacker models should be studied and formalized for databases. Rather than being implicit, the relevant models must be made explicit, just like when analysing security in other domains. In this respect, we presented a concrete attacker model that accounts for relevant features of modern databases, like triggers and views, and attacker inference capabilities.

Second, access control mechanisms must be designed to be secure, and provably so, with respect to the formalized attacker capabilities. This requires research on mechanism design, complemented by a formal operational semantics of databases as a basis for security proofs. We presented such a mechanism, proved that it is secure, and built and evaluated a prototype of it in PostgreSQL. As a future work, we will extend our framework and our PDP to directly support the SQL language, and we will investigate efficiency improvements for our PDP.

Acknowledgments. We thank Ùlfar Erlingsson, Erwin Fang, Andreas Lochbihler, Ognjen Maric, Mohammad Torabi

Dashti, Dmitriy Traytel, Petar Tsankov, Thilo Weghorn, Der-Yeuan Yu, Eugen Zalinescu, as well as the anonymous reviewers for their comments.

9. REFERENCES

[1] "New Security Features in Sybase Adaptive Server Enterprise," *Sybase Technical White Paper, Sybase, an SAP company*, 2003.

[2] (2014, Sep.) Manage trigger security, *Microsoft MSDN Library*. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms191134.aspx/

[3] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of databases*. Addison-Wesley Reading, 1995, vol. 8.

[4] R. Agrawal, P. Bird, T. Grandison, J. Kiernan, S. Logan, and W. Rjaibi, "Extending relational database systems to automatically enforce privacy policies," in *Proc. 2005 IEEE Int. Conf. Data Engineering*.

[5] A. Askarov and S. Chong, "Learning is change in knowledge: Knowledge-based security for dynamic policies," in *Proc. 2012 IEEE Symp. Computer Security Foundations*.

[6] A. Askarov and A. Sabelfeld, "Gradual release: Unifying declassification, encryption and key release policies," in *Proc. 2007 IEEE Symp. Security and Privacy*.

[7] ——, "Tight enforcement of information-release policies for dynamic languages," in *Proc. 2009 IEEE Symp. Computer Security Foundations*.

[8] G. Bender, L. Kot, and J. Gehrke, "Explainable security for relational databases," in *Proc. 2014 ACM Intl. Conf. Management of data*.

[9] G. M. Bender, L. Kot, J. Gehrke, and C. Koch, "Fine-grained disclosure control for app ecosystems," in *Proc. 2013 ACM Intl. Conf. Management of data*.

[10] A. Bohannon, B. C. Pierce, V. Sjöberg, S. Weirich, and S. Zdancewic, "Reactive noninterference," in *Proc. 2009 ACM Conf. Computer and Communications Security*.

[11] P. A. Bonatti, S. Kraus, and V. Subrahmanian, "Foundations of secure deductive databases," *IEEE Trans. Knowl. Data Eng.*, vol. 7, no. 3, 1995.

[12] A. Brodsky, C. Farkas, and S. Jajodia, "Secure databases: Constraints, inference channels, and monitoring disclosures," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 6, 2000.

[13] K. Browder and M. Davidson, "The virtual private database in oracle9ir2," *Oracle Technical White Paper, Oracle Corporation*, vol. 500, 2002.

[14] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, "The maude 2.0 system," in *Rewriting Techniques and Applications*. Springer, 2003.

[15] B. J. Corcoran, N. Swamy, and M. Hicks, "Cross-tier, label-based security enforcement for web applications," in *Proc. 2009 ACM Intl. Conf. Management of data*.

[16] B. Davis and H. Chen, "DBTaint: cross-application information flow tracking via databases," *Proc. 2010 USENIX Conf. Web Application Development*.

[17] D. E. Denning and T. F. Lunt, "A multilevel relational data model," in *Proc. 1987 IEEE Symp. Security and Privacy*.

[18] D. Devriese and F. Piessens, "Noninterference through secure multi-execution," in *Proc. 2010 IEEE Symp. Security and Privacy*.

[19] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. 29, no. 2, 1983.

[20] C. Farkas and S. Jajodia, "The inference problem: a survey," *ACM SIGKDD Explorations*, vol. 4, no. 2, 2002.

[21] R. Giacobazzi and I. Mastroeni, "Abstract non-interference: Parameterizing non-interference by abstract interpretation," in *Proc. 2004 ACM Symp. Principles of Programming Languages*.

[22] J. A. Goguen and J. Meseguer, "Security policies and security models," in *IEEE Symp. Security and Privacy*, 1982.

[23] P. P. Griffiths and B. W. Wade, "An authorization mechanism for a relational database system," *ACM Trans. on Database Syst.*, vol. 1, no. 3, 1976.

[24] M. Guarnieri and D. Basin, "Optimal security-aware query processing," in *Proc. 2014 Int. Conf. Very Large Data Bases*.

[25] M. Guarnieri, S. Marinovic, and D. Basin, "Strong and provably secure database access control," in *Proc. 2016 IEEE European Symp. Security and Privacy*.

[26] ——. Strong and Provably Secure Database Access Control — Prototype and Maude models. [Online]. Available: http://www.infsec.ethz.ch/research/projects/FDAC.html

[27] R. Halder and A. Cortesi, "Fine grained access control for relational databases by abstract interpretation," in *Software and Data Technologies*, 2013, vol. 170.

[28] D. Hedin and A. Sabelfeld, "A perspective on information-flow control," *Proc. 2011 Marktoberdorf Summer School. IOS Press*, 2011.

[29] S. Jajodia and R. Sandhu, "Polyinstantiation integrity in multilevel relations," in *Proc. 1990 IEEE Symp. Security and Privacy*.

[30] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu, "Query-based data pricing," in *Proc. 2012 ACM Symp. Principles of Database Systems*.

[31] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt, "Limiting disclosure in hippocratic databases," in *Proc. 2004 Int. Conf. Very Large Data Bases*.

[32] P. Li and S. Zdancewic, "Practical information flow control in web-based information systems," in *Proc. 2005 IEEE Workshop on Computer Security Foundations*.

[33] A. C. Myers and B. Liskov, "A decentralized model for information flow control," in *Proc. 1997 ACM Symp. Operating Systems Principles*.

[34] A. Nash, L. Segoufin, and V. Vianu, "Views and queries: Determinacy and rewriting," *ACM Trans. Database Syst.*, vol. 35, no. 3, 2010.

[35] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy, "Extending query rewriting techniques for fine-grained access control," in *Proc. 2004 ACM Int. Conf. Management of data*.

[36] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," *IEEE J. Sel. Areas*

Commun., vol. 21, no. 1, 2003.

[37] P. Samarati and S. Capitani de Vimercati, "Access Control: Policies, Models, and Mechanisms," *Springer Lecture Notes in Computer Science*, vol. 2171, 2001.

[38] R. Sandhu and F. Chen, "The multilevel relational (MLR) data model," *ACM Trans. Inf. Syst. Sec.*, vol. 1, no. 1, 1998.

[39] D. Schoepe, D. Hedin, and A. Sabelfeld, "SeLINQ: tracking information across application-database boundaries," in *Proc. 2014 ACM Intl. Conf. Functional Programming*.

[40] D. Schultz and B. Liskov, "IFDB: decentralized information flow control for databases," in *Proc. 2013 ACM European Conf. Computer Systems*.

[41] J. Shi, H. Zhu, G. Fu, and T. Jiang, "On the soundness property for sql queries of fine-grained access control in dbmss," in *Proc. 2009 IEEE/ACIS Int. Conf. Computer and Information Science*.

[42] K. Smith and M. Winslett, "Multilevel secure rules: Integrating the multilevel secure and active data models," in *Database Security VI: Status and Prospects*. North-Holland, 1993.

[43] M. Stonebraker and E. Wong, "Access control in a relational data base management system by query modification," in *Proc. 1974 ACM Annual Conference*.

[44] T. S. Toland, C. Farkas, and C. M. Eastman, "The inference problem: Maintaining maximal availability in the presence of database updates," *Computers & Security*, vol. 29, no. 1, 2010.

[45] A. Van Gelder and R. W. Topor, "Safety and translation of relational calculus," *ACM Trans. Database Syst.*, vol. 16, no. 2, pp. 235–278, May 1991.

[46] Q. Wang, T. Yu, N. Li, J. Lobo, E. Bertino, K. Irwin, and J.-W. Byun, "On the correctness criteria of fine-grained access control in relational databases," in *Proc. 2007 Int. Conf. Very Large Data Bases*.

APPENDIX

In this appendix we formalize the system's operational semantics, the attacker model, and the security properties. Furthermore, we present complete proofs of all results.

For simplicity's sake, in the following we assume, without loss of generality, that all the relational calculus formulae do not use constant symbols inside predicates. For instance, instead of the formula $\exists x. R(x, 5, 10)$, we consider the equivalent formula $\exists x, y, z. R(x, y, z) \wedge y = 5 \wedge z = 10$. Note that this does not restrict the scope of our work as all formulae can be trivially expressed without using constant symbols inside predicates.

Structure. In Appendix A, we provide a complete formalization of our system model. In Appendix B, we present all the rules defining the $\vdash_u$ relation and we prove the soundness of $\vdash_u$ with respect to the relational calculus semantics. In Appendix C, we provide the complete formalization of the $\rightsquigarrow_{auth}$ relation. In Appendix D, we formalize u-projections and the indistinguishability relation $\cong_{P,u}$. In Appendix E we formalize the access control function f_{int}, we prove that it provides database integrity, and we prove its data complexity. In Appendix F we formalize the access control function f^u_{conf}, we prove that it provides data confidentiality, and we prove its data complexity. In Appendix G we prove that the function f, which is obtained by composing the PDPs f_{int} and f^u_{conf}, provides both database integrity and data confidentiality. We also prove that its data complexity is AC^0. Finally, in Appendix H we show that the concepts of secure judgment and data confidentiality have precise interpretations in terms of non-interference.

A. FORMALIZING THE SYSTEM MODEL

In this section, we precisely formalize the system model introduced in §4.2. We first introduce some auxiliary definitions about queries, views, privileges, and triggers. Afterwards, we introduce the concept of partial state. Then, we formalize contexts and we refine the notion of M-state given in §4.2. Finally, we formalize the transition relation $\rightarrow_f$ together with some auxiliary predicates and functions.

A.1 Auxiliary definitions on queries, views, privileges, and triggers

Triggers can give rise to non-terminating executions, for example when the action associated with trigger t_1 activates trigger t_2, which in turn activates t_1. We say that a set T of triggers is *safe* iff no trigger in T can activate another trigger in T. Note that safety ensures termination. Even though this termination condition is simple, it is sufficient for the purpose of this paper. Note that more complex and permissive termination conditions do not influence our results. We say that a set of triggers T is *safe*, denoted by $\mathit{safe}(T)$, iff for all triggers $t_1, t_2 \in T$:

- if the t_1's activation event is an `INSERT` on the table R, then t_2's action is not of the form $\langle \texttt{INSERT}, R, \overline{t}\rangle$, or
- if the t_1's activation event is a `DELETE` on the table R, then t_2's action is not of the form $\langle \texttt{DELETE}, R, \overline{t}\rangle$.

Let D be a database schema, U be a set of users, t be a trigger over D, and V be a set of views over D. We say that t is a *U-trigger*, denoted by $\mathit{usersIn}(t, U)$, if and only if $\mathit{owner}(t) \in U$ and t's statement mentions just users in U. We say that *a query q is defined over D and V*, denoted by $\mathit{defined}(q, D, V)$, iff all the predicates in q are either tables in D or views in V. We say that *a privilege p is defined over D and V*, denoted by $\mathit{defined}(p, D, V)$, iff the table or view referred in p is in $D \cup V$. We say that *a view v is defined over D and V*, denoted by $\mathit{defined}(v, D, V)$, iff its definition is defined over D and V. Finally, we say that *a trigger t is defined over D and V*, denoted by $\mathit{defined}(t, D, V)$, iff (1) the table on which t is defined is in D, (2) t's `WHEN` condition is defined over D and V, and (3) t's action refers only to tables and views in $D \cup V$.

A.2 Revoke Semantics

We now define the function *revoke* that models the semantics of SQL's `REVOKE` statements with cascade. In the following, let S be a security policy, i.e., a set of `GRANT`s, u_1, u_2, u_3, u_4, u, and u' be user identifiers, $op, op' \in \{\oplus, \oplus^*\}$, and p be a privilege. We say that a *chain* is a sequence of grants $g_1 \cdot g_2 \cdot \ldots \cdot g_n$ such that (1) $g_1 = \langle op', u_4, p, \mathit{start}\rangle$, (2) if $p \neq \langle \texttt{SELECT}, V\rangle$, where V is a view with owner's privileges, then $\mathit{start} = \mathit{admin}$, whereas if $p = \langle \texttt{SELECT}, V\rangle$, then $\mathit{start} \in \{\mathit{admin}, \mathit{owner}(V)\}$, and (3) for each $1 \leq i \leq n-1$, $g_i = \langle \oplus^*, u_2, p, u_1\rangle$ and $g_{i+1} = \langle op, u_3, p, u_2\rangle$. We first define the *chain* function that takes as input a policy S and constructs all possible chains.

$$\begin{aligned}\mathit{chain}(S) :=&\{\langle op, u, p, u'\rangle \in S \mid u' = \mathit{admin}\} \cup \\ &\{\langle op, u, \langle \texttt{SELECT}, V\rangle, u'\rangle \in S \mid V = \langle v, o, q, O\rangle \\ &\quad \wedge u' = o\} \cup \\ &\bigcup_{g_1 \cdot \ldots \cdot g_n \in \mathit{chain}(S)} \{g_1 \cdot \ldots \cdot g_n \cdot g \mid g \in S \wedge \\ &\quad g = \langle op, u, p, u'\rangle \wedge g_n = \langle \oplus^*, u', p, u''\rangle \wedge \\ &\quad \forall i \in \{1, \ldots, n\}.\, g_i \neq g\}.\end{aligned}$$

The function *filter* takes as input a set of chains C and a grant g and returns as output the set of all chains in C that do not contain g:

$$\mathit{filter}(C, g) := \{g_1 \cdot \ldots \cdot g_n \in C \mid \forall i \in \{1, \ldots, n\}.\, g_i \neq g\}.$$

The function *policy* takes as input a set of chains and constructs a policy, i.e., a set of grants, out of it:

$$\mathit{policy}(C) := \bigcup_{g_1 \cdot \ldots \cdot g_n \in C} \; \bigcup_{1 \leq i \leq n} \{g_i\}.$$

Finally, the function *revoke*, which models the semantics of the `REVOKE` command with cascade, is as follows:

$$\begin{aligned}\mathit{revoke}(S, u, p, u') :=& \mathit{policy}(\mathit{filter}(\mathit{chain}(\mathit{policy}(\mathit{filter}(\\ &\mathit{chain}(S), \langle \oplus, u, p, u'\rangle))), \langle \oplus^*, u, p, u'\rangle)).\end{aligned}$$

Given a policy S, $\mathit{revoke}(S, u, p, u')$ denotes the policy obtained by applying $\langle \ominus, u, p, u'\rangle$ to S.

A.3 Partial States

An *M-partial state* is a tuple $\langle db, U, sec, T, V\rangle$ such that $db \in \Omega_D^\Gamma$ is a database state, $U \subset \mathcal{U}$ is a finite set of users such that $\mathit{admin} \in U$, $sec \in \mathcal{S}_{U,D}$ is a security policy, T is a finite set of safe triggers over D, and V is a finite set of views over D. We denote by Π_M the set of all M-partial states. Given an M-state $s = \langle db, U, sec, T, V, c\rangle$, we denote by $\mathit{pState}(s)$ the M-partial state $\langle db, U, sec, T, V\rangle$ obtained from s by dropping the context c.

A.4 Contexts

Let $M = \langle D, \Gamma\rangle$ be a system configuration and u be a user. An *(M, u)-action effect* is a tuple $\langle act, accDec, res, E\rangle$, where $act \in \mathcal{A}_{D,u}$ is an action, $accDec \in \{\top, \bot\}$ is the access control decision for that action (where $\top$ stands for *permit* and $\bot$ stands for *deny*), $res \in \{\top, \bot\}$ is the action result, and $E \subseteq \Gamma$ is the set of integrity constraints violated by the action. We denote by $\Omega_{M,u}^{\mathit{actEff}}$ the set of all (M, u)-action effects and by $\Omega_{M,U}^{\mathit{actEff}}$, for some $U \subseteq \mathcal{U}$, the set $\bigcup_{u \in U} \Omega_{M,u}^{\mathit{actEff}}$. An *$(M, u)$-trigger effect* is a triple $\langle t, \mathit{when}, \mathit{stmt}\rangle$ where $t \in \mathcal{TRIGGER}_D$ is a trigger, $\mathit{when} \in \Omega_{M,u}^{\mathit{actEff}}$ is the action effect associated with the `WHEN` condition of the trigger, and $\mathit{stmt} \in \Omega_{M,u}^{\mathit{actEff}} \cup \{\epsilon\}$ is the action effect associated with the statement in the trigger's body. We denote by $\Omega_{M,u}^{\mathit{triEff}}$ the set of all (M, u)-trigger effects and by $\Omega_{M,U}^{\mathit{triEff}}$, for some $U \subseteq \mathcal{U}$, the set $\bigcup_{u \in U} \Omega_{M,u}^{\mathit{triEff}}$.

An *M-pending trigger transaction* is a 4-tuple $\langle s, \overline{t}, u, tr\rangle$, where $s \in \Pi_M \cup \{\epsilon\}$ is an M-partial state representing the "roll-back state", i.e., the state that we must restore in case a roll-back happens, $\overline{t} \in \{\epsilon\} \cup \bigcup_{n \in \mathbb{N}^+} \mathbf{dom}^n$ is the tuple involved in the event that has fired the transaction, $u \in \mathcal{U} \cup \{\epsilon\}$ is the user that has activated the triggers in the transactions, and $tr \in \mathcal{TRIGGER}_D^*$ is a sequence of triggers. Note that we denote by $\cdot$ the concatenation operation between strings over $\mathcal{TRIGGER}_D^*$, by ϵ the empty string in $\mathcal{TRIGGER}_D^*$, and by $\langle \epsilon, \epsilon, \epsilon, \epsilon \rangle$ the empty M-pending transaction.

An *M-history* h is a sequence of action effects and trigger effects, i.e., $h \in (\Omega_{M,\mathcal{U}}^{actEff} \cup \Omega_{M,\mathcal{U}}^{triEff})^*$. We denote by $\mathcal{H}_M$ the set of all M-histories, by $\cdot$ the concatenation operation over $\mathcal{H}_M$, and by ϵ the empty history.

We are now ready to formally define contexts. Let $M = \langle D, \Gamma \rangle$ be a system configuration. An *M-context* is a tuple $\langle h, actEff, tr \rangle$, where $h \in \mathcal{H}_M$ models the system's history, $actEff \in \Omega_{M,\mathcal{U}}^{actEff} \cup \Omega_{M,\mathcal{U}}^{triEff} \cup \{\epsilon\}$ describes the effect of the last action, i.e., whether the action has been accepted by the access control mechanism and the action's result, and tr is an M-pending transaction. Furthermore, the empty context ϵ is the element $\langle \epsilon, \epsilon, \langle \epsilon, \epsilon, \epsilon, \epsilon \rangle \rangle$.

A.4.1 Auxiliary Functions over contexts

Given an M-context $c = \langle h, actEff, tr \rangle$, we denote by $secEx$ the following function, which returns $\top$ if the last action has caused a security exception.

$$secEx(\langle h, aE, tr \rangle) = \begin{cases} \top & \text{if } aE = \langle act, \bot, res, E \rangle \\ \top & \text{if } aE = \langle t, \langle act, \bot, res, E \rangle, \epsilon \rangle \\ \top & \text{if } aE = \langle t, when, \langle act, \bot, res, E \rangle \rangle \\ \bot & \text{otherwise} \end{cases}$$

Similarly, we denote by $Ex(c)$ the function extracting the integrity constraints violated by the last action.

$$Ex(\langle h, aE, tr \rangle) = \begin{cases} E & \text{if } aE = \langle act, aC, res, E \rangle \\ E & \text{if } aE = \langle t, when, \langle act, aC, res, E \rangle \rangle \\ \emptyset & \text{otherwise} \end{cases}$$

We also denote by $res(c)$ the function extracting the last action's result:

$$res(\langle h, aE, tr \rangle) = \begin{cases} res & \text{if } aE = \langle act, aC, res, E \rangle \\ aC & \text{if } aE = \langle t, \langle act, aC, res, E \rangle, \epsilon \rangle \\ aC \wedge aC' & \text{if } aE = \langle t, \langle act, aC, res, E \rangle, \\ \quad \wedge res' & \quad \langle act', aC', res', E' \rangle \rangle \wedge \\ & \quad \langle act', aC', res', E' \rangle \neq \epsilon \end{cases}$$

Similarly, we denote by $acA(c)$ and $acC(c)$ the functions that extract the access control decision for the trigger's action and condition:

$$acA(\langle h, aE, tr \rangle) = \begin{cases} aC' & \text{if } aE = \langle t, \langle act, aC, res, E \rangle, \\ & \quad \langle act', aC', res', E' \rangle \rangle \wedge \\ & \quad \langle act', aC', res', E' \rangle \neq \epsilon \\ \bot & \text{otherwise} \end{cases}$$

$$acC(\langle h, aE, tr \rangle) = \begin{cases} aC & \text{if } aE = \langle t, \langle act, aC, res, E \rangle, \epsilon \rangle \\ aC & \text{if } aE = \langle t, \langle act, aC, res, E \rangle, \\ & \quad \langle act', aC', res', E' \rangle \rangle \wedge \\ & \quad \langle act', aC', res', E' \rangle \neq \epsilon \\ \bot & \text{otherwise} \end{cases}$$

We denote by $invoker(c)$ the function extracting the user in the transaction, i.e., $invoker(\langle h, aE, \langle s, \bar{t}, u, trL \rangle \rangle) = u$. Similarly, we denote by $tpl(c)$ the function extracting the tuple that has fired the transaction, namely $tpl(\langle h, aE, \langle s, \bar{t}, u, trL \rangle \rangle) = \bar{t}$, by $triggers(c)$ the function extracting the list of triggers, i.e., $triggers(\langle h, aE, \langle s, \bar{t}, u, trL \rangle \rangle) = trL$, and by $trigger(c)$, or $tr(c)$ for short, the first trigger in the sequence $triggers(c)$.

A.5 States

We can now define M-states. Let $M = \langle D, \Gamma \rangle$ be a system configuration. An *M-state* is a tuple $\langle db, U, sec, T, V, c \rangle$ such that $db \in \Omega_D^\Gamma$ is a database state, $U \subset \mathcal{U}$ is a finite set of users such that $admin \in U$, $sec \in \mathcal{S}_{U,D}$ is a security policy, T is a finite set of safe triggers over D such that for any trigger $t \in T$, both $usersIn(t, U)$ and $defined(t, D, V)$ hold, V is a finite set of views over D such that (1) there are no cyclic dependencies between the views in V, and (2) for any view $v \in V$, $defined(t, D, V')$, for some $V' \subseteq V$, and v's owner is in U, and $c \in \mathcal{C}_M$ is an M-context.

In Section 4.2, we denoted an M-state as a tuple $\langle db, sec, U, T, V, c \rangle$, where $c = \langle h, actEff, tr \rangle$ is an element of $\mathcal{C}_M$. In the following, instead of representing states as $\langle db, sec, U, T, V, \langle h, aE, tr \rangle \rangle$, we represent them as $\langle db, sec, U, T, V, h, aE, tr \rangle$. Given an M-state $s := \langle db, sec, U, T, V, h, aE, tr \rangle$, we denote by $ctx(s)$ the context $\langle h, aE, tr \rangle$. With a slight abuse of notation, we extend the functions Ex, $secEx$, res, tpl, acA, acC, $triggers$, tr, and $invoker$ from contexts to M-states, e.g., $Ex(s)$ is just $Ex(ctx(s))$. Furthermore, given an M-state $s := \langle db, sec, U, T, V, h, aE, tr \rangle$, we use a dot notation to refer to its components. For instance, we use $s.db$ to refer to the database's state in s and $s.sec$ to refer to the policy in s.

A.6 Transition Relation $\rightarrow_f$

The transition rules describing the $\rightarrow_f$ transition relation are shown in Figures 12–19. The $\rightarrow_f$ relation is, thus, the smallest relation satisfying all the inference rules. Note that we ignore the upd function introduced in Section 4.2 since the rules explicitly encode the changes to the contexts.

We now define the functions we used in the rules in Figures 12–19. The $getActualUser(m, invk, ow)$ function, where $m \in \{A, O\}$ and $invk, ow \in \mathcal{U}$, is defined as follows:

$$getActualUser(m, invk, ow) = \begin{cases} invk & \text{if } m = A \\ ow & \text{if } m = O \end{cases}$$

The ID function takes as input an action $act \in \mathcal{A}_{D,\mathcal{U}}$ and returns $\top$ if act is either $\langle u, \texttt{INSERT}, R, \bar{t} \rangle$ or $\langle u, \texttt{DELETE}, R, \bar{t} \rangle$, for some $u \in \mathcal{U}$, $R \in D$, and $\bar{t} \in \mathbf{dom}^{|R|}$. The function ID returns $\bot$ otherwise.

The $apply$ function, which takes as input an action $act \in \mathcal{A}_{M,\mathcal{U}}$ that is either an `INSERT` or a `DELETE` action and a database state $db \in \Omega_D$, is defined as follows:

$$apply(act, db) = \begin{cases} db[R \oplus \bar{t}] & \text{if } act = \langle u, INSERT, R, \bar{t} \rangle \\ db[R \ominus \bar{t}] & \text{if } act = \langle u, DELETE, R, \bar{t} \rangle \end{cases}$$

Let $t = \langle id, ow, ev, R', \phi, stmt, m \rangle$ be a trigger and R be a relation schema. We denote t's owner by $owner(t)$, i.e., $owner(t) = ow$. Similarly, given a view V, we denote by $owner(V)$ the owner of V. We also denote by $\overline{x}^{|R|}$ the tuple of variables $\langle x_1, \ldots, x_{|R|} \rangle$. Furthermore, given a tuple $\bar{t} := \langle t_1, \ldots, t_n \rangle$, we denote by $\bar{t}(i)$ the i-th value t_i. Finally, we denote by $\bar{t}[\overline{x}^{|R|} \mapsto \overline{v}]$, where $\bar{t}$ is a tuple of values in $\mathbf{dom}$ and variables in $\{x_1, \ldots, x_{|R|}\}$ and $\overline{v}$ is a tuple in $\mathbf{dom}^{|R|}$, the tuple $\overline{z} \in \mathbf{dom}^n$ obtained as follows: for each $i \in \{1, \ldots, n\}$, if $\bar{t}(i) = x_j$, where $x_j \in \{x_1, \ldots, x_{|R|}\}$, then $\overline{z}(i) = \overline{v}(j)$, and otherwise $\overline{z}(i) = \bar{t}(i)$. We are now ready to define the function $getAction$, which takes as input the trigger's statement $stmt$, a user u, and a tuple $\bar{t}' \in \mathbf{dom}^{|R'|}$, and returns the concrete action executed by the system. Formally, $getAction$ is as follows:

- $getAction(\langle \texttt{INSERT}, R, \bar{t}\rangle, u, \bar{t}') = \langle u, \texttt{INSERT}, R, \bar{t}[\overline{x}^{|R'|} \mapsto \overline{t'}]\rangle$,
- $getAction(\langle \texttt{DELETE}, R, \bar{t}\rangle, u, \bar{t}') = \langle u, \texttt{DELETE}, R, \bar{t}[\overline{x}^{|R'|} \mapsto \overline{t'}]\rangle$, and
- $getAction(\langle op, u, p\rangle, u, \bar{t}') = \langle op, u, p, u\rangle$, where $op \in \{\ominus, \oplus, \oplus^*\}$.

We assume there is a total-order relation $\preceq_{\mathcal{T}}$ over $\mathcal{T}$. We use this ordering to determine the order in which triggers are executed. Given a set of triggers T and a database schema D, we denote by $\mathit{filter}(T, ev, R)$, where $ev \in \{INS, DEL\}$ and $R \in D$, the sequence of triggers in T (ordered according to $\preceq_{\mathcal{T}}$) whose event is ev and whose relation schema is R.

$$\frac{admin \in U \quad aE' = \langle\langle admin, \texttt{ADD\_USER}, u\rangle, \top, \top, \emptyset\rangle \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, \langle admin, \texttt{ADD\_USER}, u\rangle) = \top}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle admin, \texttt{ADD\_USER}, u\rangle}_f \langle db, U \cup \{u\}, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}\ \text{Add User Success}$$

$$\frac{admin \in U \quad aE' = \langle\langle admin, \texttt{ADD\_USER}, u\rangle, \bot, \bot, \emptyset\rangle \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, \langle admin, \texttt{ADD\_USER}, u\rangle) = \bot}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle admin, \texttt{ADD\_USER}, u\rangle}_f \langle db, U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}\ \text{Add User Deny}$$

$$\frac{\begin{array}{c} u \in U \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, \langle u, \texttt{SELECT}, q\rangle) = \top \quad [q]^{db} = v \\ aE' = \langle\langle u, \texttt{SELECT}, q\rangle, \top, v, \emptyset\rangle \qquad defined(q, D, V) \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{SELECT}, q\rangle}_f \langle db, U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}\ \texttt{SELECT}\ \text{Success}$$

$$\frac{\begin{array}{c} u \in U \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, \langle u, \texttt{SELECT}, q\rangle) = \bot \\ aE' = \langle\langle u, \texttt{SELECT}, q\rangle, \bot, \bot, \emptyset\rangle \qquad defined(q, D, V) \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{SELECT}, q\rangle}_f \langle db, U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}\ \texttt{SELECT}\ \text{Deny}$$

Figure 12: Rules defining the $\rightarrow_f$ relation for SELECT and ADD USER

$$\frac{\begin{array}{c} u \in U \quad R \in D \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, act) = \top \quad act = \langle u, \texttt{INSERT}, R, \bar{t}\rangle \\ db[R \oplus \bar{t}] \in \Omega_D^\Gamma \qquad aE' = \langle act, \top, \top, \emptyset\rangle \qquad filter(T, INS, R) = \epsilon \vee \bar{t} \in db(R) \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{INSERT}, R, \bar{t}\rangle}_f \langle db[R \oplus \bar{t}], U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}\ \texttt{INSERT}\ \text{Success 1}$$

$$\frac{\begin{array}{c} u \in U \quad R \in D \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, act) = \top \quad act = \langle u, \texttt{INSERT}, R, \bar{t}\rangle \quad db[R \oplus \bar{t}] \in \Omega_D^\Gamma \\ aE' = \langle act, \top, \top, \emptyset\rangle \qquad tr = filter(T, INS, R) \qquad tr \neq \epsilon \qquad \bar{t} \notin db(R) \qquad rS' = \langle db, U, sec, T, V\rangle \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{INSERT}, R, \bar{t}\rangle}_f \langle db[R \oplus \bar{t}], U, sec, T, V, h \cdot aE, aE', \langle rS', \bar{t}, u, tr\rangle\rangle}\ \texttt{INSERT}\ \text{Success 2}$$

$$\frac{\begin{array}{c} u \in U \quad R \in D \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, \langle u, \texttt{INSERT}, R, \bar{t}\rangle) = \top \\ E' = \{\phi \in \Gamma | [\phi]^{db[R \oplus \bar{t}]} = \bot\} \qquad E' \neq \emptyset \qquad aE' = \langle\langle u, \texttt{INSERT}, R, \bar{t}\rangle, \top, \bot, E'\rangle \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{INSERT}, R, \bar{t}\rangle}_f \langle db, U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}\ \texttt{INSERT}\ \text{Exception}$$

$$\frac{u \in U \quad R \in D \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, \langle u, \texttt{INSERT}, R, \bar{t}\rangle) = \bot \quad aE' = \langle\langle u, \texttt{INSERT}, R, \bar{t}\rangle, \bot, \bot, \emptyset\rangle}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{INSERT}, R, \bar{t}\rangle}_f \langle db, U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}\ \texttt{INSERT}\ \text{Deny}$$

Figure 13: Rules defining the $\rightarrow_f$ relation for INSERT

$$\frac{\begin{array}{c} u \in U \quad R \in D \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, act) = \top \quad act = \langle u, \texttt{DELETE}, R, \bar{t}\rangle \\ db[R \ominus \bar{t}] \in \Omega_D^\Gamma \qquad aE' = \langle act, \top, \top, \emptyset\rangle \qquad filter(T, DEL, R) = \epsilon \vee \bar{t} \notin db(R) \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{DELETE}, R, \bar{t}\rangle}_f \langle db[R \ominus \bar{t}], U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}\ \texttt{DELETE}\ \text{Success 1}$$

$$\frac{\begin{array}{c} u \in U \quad R \in D \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, act) = \top \quad act = \langle u, \texttt{DELETE}, R, \bar{t}\rangle \quad db[R \ominus \bar{t}] \in \Omega_D^\Gamma \\ aE' = \langle act, \top, \top, \emptyset\rangle \qquad tr = filter(T, DEL, R) \qquad tr \neq \epsilon \qquad \bar{t} \in db(R) \qquad rS' = \langle db, U, sec, T, V\rangle \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{DELETE}, R, \bar{t}\rangle}_f \langle db[R \ominus \bar{t}], U, sec, T, V, h \cdot aE, aE', \langle rS', \bar{t}, u, tr\rangle\rangle}\ \texttt{DELETE}\ \text{Success 2}$$

$$\frac{\begin{array}{c} u \in U \quad R \in D \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, \langle u, \texttt{DELETE}, R, \bar{t}\rangle) = \top \\ E' = \{\phi \in \Gamma | [\phi]^{db[R \ominus \bar{t}]} = \bot\} \qquad E' \neq \emptyset \qquad aE' = \langle\langle u, \texttt{DELETE}, R, \bar{t}\rangle, \top, \bot, E'\rangle \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{DELETE}, R, \bar{t}\rangle}_f \langle db, U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}\ \texttt{DELETE}\ \text{Exception}$$

$$\frac{u \in U \quad R \in D \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle, \langle u, \texttt{DELETE}, R, \bar{t}\rangle) = \bot \quad aE' = \langle\langle u, \texttt{DELETE}, R, \bar{t}\rangle, \bot, \bot, \emptyset\rangle}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}', u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{DELETE}, R, \bar{t}\rangle}_f \langle db, U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}\ \texttt{DELETE}\ \text{Deny}$$

Figure 14: Rules defining the $\rightarrow_f$ relation for DELETE

$$\frac{\begin{array}{c} invk, ow \in U \quad t = \langle id, ow, ev, R', \phi, stmt, m\rangle \quad u = getActualUser(m, ow, invk) \quad \phi' = \phi[\overline{x}^{|R'|} \mapsto \bar{t}] \\ f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle, \langle u, \texttt{SELECT}, \phi'\rangle) = \top \quad [\phi']^{db} = \top \quad aE' = \langle\langle u, \texttt{SELECT}, \phi'\rangle, \top, \top, \emptyset\rangle \\ act = getAction(stmt, u, \bar{t}) \quad db' = apply(act, db) \quad f(\langle db, U, sec, T, V, h \cdot aE, aE', \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle, act) = \top \\ db' \in \Omega_D^\Gamma \quad aE'' = \langle act, \top, \top, \emptyset\rangle \quad tE' = \langle t, aE', aE''\rangle \quad ID(act) = \top \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle \xrightarrow{t}_f \langle db', U, sec, T, V, h \cdot aE, tE', \langle rS, \bar{t}, invk, tr\rangle\rangle}$$

Trigger `DELETE-INSERT` Success

$$\frac{\begin{array}{c} invk, ow \in U \quad t = \langle id, ow, ev, R', \phi, stmt, m\rangle \quad u = getActualUser(m, ow, invk) \quad rS = \langle db', U', sec', T', V'\rangle \\ f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle, \langle u, \texttt{SELECT}, \phi'\rangle) = \top \quad [\phi']^{db} = \top \quad aE' = \langle\langle u, \texttt{SELECT}, \phi'\rangle, \top, \top, \emptyset\rangle \\ act = getAction(stmt, u, \bar{t}) \quad f(\langle db, U, sec, T, V, h \cdot aE, aE', \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle, act) = \top \quad ID(act) = \top \\ \phi' = \phi[\overline{x}^{|R'|} \mapsto \bar{t}] \quad E' = \{\phi \in \Gamma | [\phi]^{apply(act, db)}\} \quad E' \neq \emptyset \quad aE'' = \langle act, \top, \bot, E'\rangle \quad tE' = \langle t, aE', aE''\rangle \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle \xrightarrow{t}_f \langle db', U', sec', T', V', h \cdot aE, tE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}$$

Trigger `DELETE-INSERT` Exception

Figure 15: Rules defining the $\to_f$ relation for triggers with `INSERT`/`DELETE` action

$$\frac{\begin{array}{c} invk, ow \in U \quad t = \langle id, ow, ev, R', \phi, stmt, m\rangle \quad u = getActualUser(m, ow, invk) \quad \phi' = \phi[\overline{x}^{|R'|} \mapsto \bar{t}] \\ f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle, \langle u, \texttt{SELECT}, \phi'\rangle) = \top \quad [\phi']^{db} = \top \quad aE' = \langle\langle u, \texttt{SELECT}, \phi'\rangle, \top, \top, \emptyset\rangle \\ \langle op, u', p, u\rangle = getAction(stmt, u, \bar{t}) \quad f(\langle db, U, sec, T, V, h \cdot aE, aE', \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle, \langle op, u', p, u\rangle) = \top \\ aE'' = \langle\langle op, u', p, u\rangle, \top, \top, \emptyset\rangle \quad tE' = \langle t, aE', aE''\rangle \quad op \in \{\oplus, \oplus^*\} \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle \xrightarrow{t}_f \langle db, U, sec \cup \{\langle op, u', p, u\rangle\}, T, V, h \cdot aE, tE', \langle rS, \bar{t}, invk, tr\rangle\rangle}$$

Trigger `GRANT` Success

$$\frac{\begin{array}{c} invk, ow \in U \quad t = \langle id, ow, ev, R', \phi, stmt, m\rangle \quad u = getActualUser(m, ow, invk) \quad \phi' = \phi[\overline{x}^{|R'|} \mapsto \bar{t}] \\ f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle, \langle u, \texttt{SELECT}, \phi'\rangle) = \top \quad [\phi']^{db} = \top \quad aE' = \langle\langle u, \texttt{SELECT}, \phi'\rangle, \top, \top, \emptyset\rangle \\ \langle \ominus, u', p, u\rangle = getAction(stmt, u, \bar{t}) \quad f(\langle db, U, sec, T, V, h \cdot aE, aE', \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle, \langle \ominus, u', p, u\rangle) = \top \\ aE'' = \langle\langle \ominus, u', p, u\rangle, \top, \top, \emptyset\rangle \quad tE' = \langle t, aE', aE''\rangle \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle \xrightarrow{t}_f \langle db, U, revoke(sec, u, p, u'), T, V, h \cdot aE, tE', \langle rS, \bar{t}, invk, tr\rangle\rangle}$$

Trigger `REVOKE` Success

Figure 16: Rules defining the $\to_f$ relation for triggers with `GRANT`/`REVOKE` action

$$\frac{\begin{array}{c} invk, ow \in U \quad t = \langle id, ow, ev, R', \phi, stmt, m\rangle \quad u = getActualUser(m, ow, invk) \\ \phi' = \phi[\overline{x}^{|R'|} \mapsto \bar{t}] \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, tr\rangle\rangle, \langle u, \texttt{SELECT}, \phi'\rangle) = \top \\ [\phi']^{db} = \bot \quad aE' = \langle\langle u, \texttt{SELECT}, \phi'\rangle, \top, \bot, \emptyset\rangle \quad tE' = \langle t, aE', \epsilon\rangle \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle \xrightarrow{t}_f \langle db, U, sec, T, V, h \cdot aE, tE', \langle rS, \bar{t}, invk, tr\rangle\rangle}$$

Trigger Disabled

$$\frac{\begin{array}{c} invk, ow \in U \quad t = \langle id, ow, ev, R', \phi, stmt, m\rangle \quad u = getActualUser(m, ow, invk) \\ rS = \langle db', U', sec', T', V'\rangle \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, tr\rangle\rangle, \langle u, \texttt{SELECT}, \phi'\rangle) = \bot \\ aE' = \langle\langle u, \texttt{SELECT}, \phi'\rangle, \bot, \bot, \emptyset\rangle \quad tE' = \langle t, aE', \epsilon\rangle \quad \phi' = \phi[\overline{x}^{|R'|} \mapsto \bar{t}] \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle \xrightarrow{t}_f \langle db', U', sec', T', V', h \cdot aE, tE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}$$

Trigger Deny Condition

$$\frac{\begin{array}{c} invk, ow \in U \quad t = \langle id, ow, ev, R', \phi, stmt, m\rangle \quad u = getActualUser(m, ow, invk) \\ f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle, \langle u, \texttt{SELECT}, \phi'\rangle) = \top \quad [\phi']^{db} = \top \quad aE' = \langle\langle u, \texttt{SELECT}, \phi'\rangle, \top, \top, \emptyset\rangle \\ act = getAction(stmt, u, \bar{t}) \quad f(\langle db, U, sec, T, V, h \cdot aE, aE', \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle, act) = \bot \quad \phi' = \phi[\overline{x}^{|R'|} \mapsto \bar{t}] \\ aE'' = \langle act, \bot, \bot, \emptyset\rangle \quad tE' = \langle t, aE', aE''\rangle \quad rS = \langle db', U', sec', T', V'\rangle \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, invk, t \cdot tr\rangle\rangle \xrightarrow{t}_f \langle db', U', sec', T', V', h \cdot aE, tE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}$$

Trigger Deny Action

Figure 17: Rules defining the $\to_f$ relation for triggers

$$\frac{\begin{array}{c} u, u' \in U \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u'', \epsilon\rangle\rangle, \langle op, u, p, u'\rangle) = \top \\ aE' = \langle\langle op, u, p, u'\rangle, \top, \top, \emptyset\rangle \quad op \in \{\oplus, \oplus^*\} \quad defined(p, D, V) \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u'', \epsilon\rangle\rangle \xrightarrow{\langle op, u, p, u'\rangle}_f \langle db, U, sec \cup \{\langle op, u, p, u'\rangle\}, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}$$

`GRANT` Success

$$\frac{\begin{array}{c} u, u' \in U \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u'', \epsilon\rangle\rangle, \langle \ominus, u, p, u'\rangle) = \top \\ aE' = \langle\langle \ominus, u, p, u'\rangle, \top, \top, \emptyset\rangle \quad defined(p, D, V) \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u'', \epsilon\rangle\rangle \xrightarrow{\langle \ominus, u, p, u'\rangle}_f \langle db, U, revoke(sec, u, p, u'), T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}$$

`REVOKE` Success

$$\frac{\begin{array}{c} u, u' \in U \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u'', \epsilon\rangle\rangle, \langle op, u, p, u'\rangle) = \bot \\ aE' = \langle\langle op, u, p, u'\rangle, \bot, \bot, \emptyset\rangle \quad op \in \{\oplus, \oplus^*, \ominus\} \quad defined(p, D, V) \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u'', \epsilon\rangle\rangle \xrightarrow{\langle op, u, p, u'\rangle}_f \langle db, U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}$$

`GRANT-REVOKE` Deny

Figure 18: Rules defining the $\to_f$ relation for `GRANT` and `REVOKE`

$$\frac{\begin{array}{c} u \in U \quad \mathit{defined}(t, D, V) \quad \mathit{safe}(\{t\} \cup T) \quad \mathit{usersIn}(t, U) \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u', \epsilon \rangle\rangle, \langle u, \texttt{CREATE}, t\rangle) = \top \\ t = \langle id, u, ev, R, \phi, stmt, m\rangle \qquad aE' = \langle\langle u, \texttt{CREATE}, t\rangle, \top, \top, \emptyset\rangle \qquad \neg\exists t' \in T.\, t' = \langle id, ow', ev', R', \phi', stmt', m'\rangle \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{CREATE}, t\rangle}_f \langle db, U, sec, T \cup \{t\}, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle} \begin{array}{c}\texttt{CREATE} \\ \texttt{TRIGGER} \\ \text{Success}\end{array}$$

$$\frac{\begin{array}{c} u \in U \quad \mathit{defined}(t, D, V) \quad \mathit{safe}(\{t\} \cup T) \quad \mathit{usersIn}(t, U) \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u', \epsilon \rangle\rangle, \langle u, \texttt{CREATE}, t\rangle) = \top \\ t = \langle id, u, ev, R, \phi, stmt, m\rangle \quad aE' = \langle\langle u, \texttt{CREATE}, t\rangle, \top, \bot, \emptyset\rangle \quad t' = \langle id, ow', ev', R', \phi', stmt', m'\rangle \quad t' \in T \quad t' \neq t \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{CREATE}, t\rangle}_f \langle db, U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle} \begin{array}{c}\texttt{CREATE} \\ \texttt{TRIGGER} \\ \text{Deny}\end{array}$$

$$\frac{\begin{array}{c} u \in U \quad \mathit{defined}(v, D, V) \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u', \epsilon \rangle\rangle, \langle u, \texttt{CREATE}, v\rangle) = \top \\ v = \langle id, u, q, m\rangle \qquad aE' = \langle\langle u, \texttt{CREATE}, v\rangle, \top, \top, \emptyset\rangle \qquad \neg\exists v' \in V.\, v' = \langle id, ow', q', m'\rangle \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{CREATE}, v\rangle}_f \langle db, U, sec, T, V \cup \{v\}, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle} \begin{array}{c}\texttt{CREATE} \\ \texttt{VIEW} \\ \text{Success}\end{array}$$

$$\frac{\begin{array}{c} u \in U \qquad \mathit{defined}(v, D, V) \qquad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u', \epsilon \rangle\rangle, \langle u, \texttt{CREATE}, v\rangle) = \top \\ v = \langle id, u, q, m\rangle \quad aE' = \langle\langle u, \texttt{CREATE}, v\rangle, \top, \bot, \emptyset\rangle \quad v' = \langle id, ow', q', m'\rangle \quad v' \in V \quad v \neq v' \end{array}}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{CREATE}, v\rangle}_f \langle db, U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle} \begin{array}{c}\texttt{CREATE} \\ \texttt{VIEW} \\ \text{Deny}\end{array}$$

$$\frac{u \in U \quad \mathit{defined}(o, D, V) \quad f(\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u', \epsilon \rangle\rangle, \langle u, \texttt{CREATE}, o\rangle) = \bot \quad aE' = \langle\langle u, \texttt{CREATE}, o\rangle, \bot, \bot, \emptyset\rangle}{\langle db, U, sec, T, V, h, aE, \langle rS, \bar{t}, u', \epsilon\rangle\rangle \xrightarrow{\langle u, \texttt{CREATE}, o\rangle}_f \langle db, U, sec, T, V, h \cdot aE, aE', \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle} \begin{array}{c}\texttt{CREATE} \\ \text{Deny}\end{array}$$

Figure 19: Rules defining the $\rightarrow_f$ relation for `CREATE` triggers and views

B. ATTACKER MODEL

In this section, we formalize our attacker model $\mathcal{ATK}_u$. Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP, L be the P-LTS, and $u \in \mathcal{U}$ be a user. The set $\mathcal{ATK}_u$ is the smallest set of judgments satisfying the inference rules in Figures 21–33. With a slight abuse of notation, in the rules we use $r, i \vdash_u \phi$ to denote that this judgment holds in $\mathcal{ATK}_u$, i.e., $r, i \vdash_u \phi \in \mathcal{ATK}_u$. Note that we redefine here also the rules we presented before in Figure 4.

In the rules, we use $\models_{fin}$ to denote the standard semantic entailment relation for first-order logic over finite models. We also denote by $replace(\psi, o)$, where ψ is a sentence and o is a view $\langle V, ow, \{\overline{x}|\phi\}, m\rangle \in \mathcal{VIEW}_D$, the formula ψ' obtained from ψ by replacing all occurrences of $V(\overline{x})$ with $\phi(\overline{x})$. Note that ψ and $replace(\psi, o)$ are semantically equivalent. Finally, given a database schema D, a state $s = \langle db, U, sec, T, V, ctx\rangle$, and an action $a \in \mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D$, we denote by $user(s, a)$ the following function:

$$user(s,a) = \begin{cases} invoker(s) & \text{if } tr(s) \neq \epsilon \\ u & \text{if } tr(s) = \epsilon \wedge u \in \mathcal{U} \wedge a \in \mathcal{A}_{D,u} \end{cases}$$

In the rules, we omit some details when dealing with integrity constraints. For instance, when we refer to functional dependencies of the form $\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. (R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}'$, we implicitly assume that $|\overline{y}| = |\overline{y}'|$ and $|\overline{z}| = |\overline{z}'|$. Furthermore, when we refer to tuples in R, we use the notation $(\overline{v}, \overline{w}, \overline{q})$ and we implicitly assume that $|\overline{v}| = |\overline{x}|$, $|\overline{w}| = |\overline{y}|$, and $|\overline{q}| = |\overline{z}|$. We make similar simplifications for the inclusion dependencies.

In our attacker model, we consider a very simple syntactic criterion for revising believes. Intuitively, the attacker is able to propagate the knowledge of a sentence ϕ after (or before) an `INSERT` or a `DELETE` action on a table R iff the predicate R does not occur in ϕ. We formalize this using the function $reviseBelief : traces(L) \times RC_{bool} \times traces(L) \to \{\top, \bot\}$. In Figure 20, we give the definition for the function only for the inputs r', ϕ, r such that $\phi \in RC_{bool}$ is a sentence and $r = r' \cdot act \cdot s$, where $act \in \mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D$ and $s \in \Omega_M$. If this is not the case, then $reviseBelif(r', \phi, r) = \bot$. Note that the function $tables$ takes as input a formula ϕ and returns as output the set of all tables mentioned in ϕ', where ϕ' is the formula obtained from ϕ by replacing all views with their definitions. We remark that, given a formula ϕ, if $R \notin tables(\phi)$, then the value of ϕ is independent on R, i.e., R does not determine ϕ.

In Lemma $B.1$, we prove that our attacker model is *sound* with respect to the relational calculus semantics, i.e., every judgment $r, i \vdash_u \phi$ that holds in $\mathcal{ATK}_u$ is such that ϕ is satisfied in the i-th state of r. We first introduce the concept of *derivation*. Given a judgment $r, i \vdash_u \phi$, a *derivation of $r, i \vdash_u \phi$ with respect to $\mathcal{ATK}_u$*, or *a derivation of* $r, i \vdash_u \phi$ for short, is a proof tree, obtained by applying the rules defining $\mathcal{ATK}_u$, that ends in $r, i \vdash_u \phi$. With a slight abuse of notation, we use $r, i \vdash_u \phi$ to denote both the judgment and its derivation. The length of a derivation, denoted $|r, i \vdash_u \phi|$, is the number of rule applications in it. Note that a judgments $r, i \vdash_u \phi$ holds in $\mathcal{ATK}_u$ iff there is a derivation for it.

Lemma B.1. *Let P be an extended configuration, L be the P-LTS, u be a user, $r \in traces(L)$ be an L run, $\phi \in RC_{bool}$ be a sentence, and $1 \leq i \leq |r|$. If $r, i \vdash_u \phi$ holds in $\mathcal{ATK}_u$, then $[\phi]^{db} = \top$, where $last(r^i) = \langle db, U, sec, T, V, c\rangle$.*

Proof. Let P be an extended configuration, L be the P-LTS, u be a user, $r \in traces(L)$ be an L run, $\phi \in RC_{bool}$ be a sentence, and $1 \leq i \leq |r|$. Furthermore, let $r, i \vdash_u \phi$ be a judgment that holds, i.e., there is a derivation d that ends on this judgment. We prove our claim by induction on the length of d.

Base Case: The base case is a derivation of length 1. Thus, there are a number of cases depending on the rule used to obtain $r, i \vdash_u \phi$.

1. *`SELECT` Success - 1.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{SELECT}, \phi\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db, U, sec, T, V, c'\rangle$. From the rules, it follows that $res(s) = \top$. From this and the LTS rules, it follows that $[\phi]^{db} = \top$.
2. *`SELECT` Success - 2.* The proof for this case is similar to that of *`SELECT` Success - 1*.
3. *`INSERT` Success.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $R(\overline{t})$. From the LTS rules, it follows that $db = db'[R \oplus \overline{t}]$. From $\oplus$'s definition, it follows that $\overline{t} \in db(R)$. Therefore, from the RC's semantics, it follows that $[R(\overline{t})]^{db} = \top$. Since $\phi := R(\overline{t})$, it follows that $[\phi]^{db} = \top$.
4. *`INSERT` Success - FD.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $\neg\exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. We claim that $[\phi]^{db'}$ holds. From this claim and the LTS semantics, it follows that there is no tuple $(\overline{v}', \overline{w}', \overline{q}')$ in $db'(R)$ such that $\overline{v}' = \overline{v}$ and $\overline{w}' \neq \overline{w}$. There are two cases:
   (a) The `INSERT` command causes an integrity exception, i.e., $Ex(s) \neq \emptyset$. From this and the LTS semantics, it follows that $db = db'$. From this and $[\phi]^{db'}$ holds, it follows that also $[\phi]^{db}$ holds.
   (b) The `INSERT` command does not cause any integrity exception, i.e., $Ex(s) = \emptyset$. From this, $[\phi]^{db'} = \top$,

$$
reviseBelief(p', \phi, p' \cdot act \cdot s)) = \begin{cases} \top & \text{if } act = \langle u, op, R, \overline{t}\rangle \wedge R \notin tables(\phi) \wedge op \in \{\texttt{INSERT}, \texttt{DELETE}\} \\ \top & \text{if } act = \langle id, ow, ev, R', \phi, \langle op, R, \overline{t}\rangle, m\rangle \wedge R \notin tables(\phi) \wedge op \in \{\texttt{INSERT}, \texttt{DELETE}\} \\ \top & \text{if } act = \langle id, ow, ev, R, \phi, \langle op, u, p\rangle, m\rangle \wedge op \in \{\oplus, \oplus^*, \ominus\} \\ \bot & \text{otherwise} \end{cases}
$$

Figure 20: Belief Revision

and $db(R) = db'(R) \cup \{(\overline{v}, \overline{w}, \overline{q})\}$, it follows that there is no tuple $(\overline{v}', \overline{w}', \overline{q}')$ in $db(R)$ such that $\overline{v}' = \overline{v}$ and $\overline{w}' \neq \overline{w}$. From this, it follows that also $[\phi]^{db}$ holds.

We now prove our claim that $[\phi]^{db'}$ holds. Assume, for contradiction's sake, that this is not the case. This means that there is a tuple $(\overline{v}', \overline{w}', \overline{q}')$ in $db'(R)$ such that $\overline{v}' = \overline{v}$ and $\overline{w}' \neq \overline{w}$. Let db'' be the state $db'[R \oplus (\overline{v}, \overline{w}, \overline{q})]$. From $db'' = db'[R \oplus (\overline{v}, \overline{w}, \overline{q})]$, and the fact that there is a tuple $(\overline{v}', \overline{w}', \overline{q}')$ in $db'(R)$ such that $\overline{v}' = \overline{v}$ and $\overline{w}' \neq \overline{w}$, it follows that there are two tuples $(\overline{v}, \overline{w}, \overline{q})$ and $(\overline{v}, \overline{w}', \overline{q}')$ in $db''(R)$ such that $\overline{w}' \neq \overline{w}$. From this and the relational calculus semantics, it follows that $[\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. ((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}']^{db''} = \bot$. This contradicts the fact that $\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. ((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}'$ is not in $Ex(s)$.

5. *INSERT Success - ID.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w})\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $\exists \overline{y}. S(\overline{v}, \overline{y})$. We claim that $[\phi]^{db'}$ holds. From this claim and the LTS semantics, it follows that there is a tuple $(\overline{v}', \overline{w}')$ in $db'(S)$ such that $\overline{v}' = \overline{v}$. There are two cases:
   (a) The INSERT command causes an integrity exception, i.e., $Ex(s) \neq \emptyset$. From this and the LTS semantics, it follows that $db = db'$. From this and $[\phi]^{db'}$ holds, it follows that also $[\phi]^{db}$ holds.
   (b) The INSERT command does not cause any integrity exception, i.e., $Ex(s) = \emptyset$. From this, $[\phi]^{db'} = \top$, and $db(S) = db'(S)$, it follows that there a tuple $(\overline{v}', \overline{w}')$ in $db(S)$ such that $\overline{v}' = \overline{v}$. From this, it follows that also $[\phi]^{db}$ holds.

   We now prove our claim that $[\phi]^{db'}$ holds. Assume, for contradiction's sake, that this is not the case. This means that there is no tuple $(\overline{v}', \overline{w}')$ in $db'(S)$ such that $\overline{v}' = \overline{v}$. Let db'' be the state $db'[R \oplus (\overline{v}, \overline{w})]$. From $db'' = db'[R \oplus (\overline{v}, \overline{w})]$, and the fact that there is no tuple $(\overline{v}', \overline{w}')$ in $db'(S)$ such that $\overline{v}' = \overline{v}$, it follows that there is a tuple $(\overline{v}, \overline{w})$ in $db''(R)$ and there is no tuple $(\overline{v}', \overline{w}')$ in $db''(S)$ such that $\overline{v}' = \overline{v}$. From this and the relational calculus semantics, it follows that $[\forall \overline{x}, \overline{z}. (R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w}))]^{db''} = \bot$. This contradicts the fact that $\forall \overline{x}, \overline{z}. (R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w}))$ is not in $Ex(s)$.
6. *DELETE Success.* The proof for this case is similar to that of *INSERT Success*.
7. *DELETE Success - ID.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{DELETE}, R, (\overline{v}, \overline{w})\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $\forall \overline{x}, \overline{z}. (S(\overline{x}, \overline{z}) \Rightarrow \overline{x} \neq \overline{v}) \vee \exists \overline{y}. (R(\overline{v}, \overline{y}) \wedge \overline{y} \neq \overline{w})$. We claim that $[\phi]^{db}$ holds. From this claim and the LTS semantics, it follows that there are two cases:
   (a) all tuples $(\overline{x}, \overline{y}) \in db(S)$ are such that $\overline{v} \neq \overline{x}$. There are two cases:
      i. The DELETE command causes an integrity exception, i.e., $Ex(s) \neq \emptyset$. From this and the LTS semantics, it follows that $db = db'$. From this and $[\phi]^{db'}$ holds, it follows that also $[\phi]^{db}$ holds.
      ii. The DELETE command does not cause any integrity exception, i.e., $Ex(s) = \emptyset$. From this, $[\phi]^{db'} = \top$, and $db(S) = db'(S)$, it follows that all tuples $(\overline{x}, \overline{y}) \in db(S)$ are such that $\overline{v} \neq \overline{x}$. Therefore, also $[\phi]^{db}$ holds.
   (b) there is a tuple $(\overline{v}, \overline{w}') \in db(R)$ such that $\overline{w} \neq \overline{w}'$. There are two cases:
      i. The DELETE command causes an integrity exception, i.e., $Ex(s) \neq \emptyset$. From this and the LTS semantics, it follows that $db = db'$. From this and $[\phi]^{db'}$ holds, it follows that also $[\phi]^{db}$ holds.
      ii. The DELETE command does not cause any integrity exception, i.e., $Ex(s) = \emptyset$. From this, $[\phi]^{db'} = \top$, and $db(R) = db'(R) \setminus \{(\overline{v}, \overline{w})\}$, it follows that there is a tuple $(\overline{v}, \overline{w}') \in db(R)$ such that $\overline{w} \neq \overline{w}'$. Therefore, also $[\phi]^{db}$ holds.

   We now prove our claim that $[\phi]^{db'}$ holds. Assume, for contradiction's sake, that this is not the case. This means that there is a tuple $(\overline{v}, \overline{z})$ in $db'(S)$ and there is no tuple $(\overline{v}, \overline{y}) \in db'(R)$ such that $\overline{y} \neq \overline{w}$. Let db'' be the state $db'[R \ominus (\overline{v}, \overline{w})]$. From $db'' = db'[R \ominus (\overline{v}, \overline{w})]$, and the fact that there is a tuple $(\overline{v}, \overline{z})$ in $db'(S)$ and there is no tuple $(\overline{v}, \overline{y}) \in db'(R)$ such that $\overline{y} \neq \overline{w}$, it follows that there is a tuple $(\overline{v}, \overline{z})$ in $db''(S)$ and there is no tuple $(\overline{v}, \overline{y}) \in db''(R)$ such that $\overline{y} \neq \overline{w}$. From this and the relational calculus semantics, it follows that $[\forall \overline{x}, \overline{z}. (S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. R(\overline{x}, \overline{w}))]^{db''} = \bot$. This contradicts the fact that $\forall \overline{x}, \overline{z}. (S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. R(\overline{x}, \overline{w}))$ is not in $Ex(s)$.
8. *INSERT Exception.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSER}, R, \overline{t}\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $\neg R(\overline{t})$. We claim that $[\neg R(\overline{t})]^{db'} = \top$ holds. From the LTS semantics, it follows that $db = db'$. Therefore, also $[\neg R(\overline{t})]^{db} = \top$ holds.
   We now prove our claim. Assume, for contradiction's sake, that $[\neg R(\overline{t})]^{db'} = \bot$. Therefore, $\overline{t} \in db'(R)$. From this and the definition of $\oplus$, it follows that $db' = db'[R \oplus \overline{t}]$. From the rules, it follows that $Ex(s) \neq \emptyset$. Therefore, from the LTS semantics, it follows that $db'[R \oplus \overline{t}] \notin \Omega_D^\Gamma$. From $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, it follows that $db' \in \Omega_D^\Gamma$. However, from $db' = db'[R \oplus \overline{t}]$ and $db' \in \Omega_D^\Gamma$, it follows that $db'[R \oplus \overline{t}] \in \Omega_D^\Gamma$ leading to a contradiction.
9. *DELETE Exception.* The proof for this case is similar to that of *INSERT Exception.*
10. *INSERT FD Exception.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be

$\exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. We claim that $[\phi]^{db'}$ holds. From this claim and the LTS semantics, it follows that there is a tuple $(\overline{v}, \overline{w}', \overline{q}')$ in $db'(R)$ such that $\overline{w}' \neq \overline{w}$. From this and $db = db'$, it follows that there is a tuple $(\overline{v}, \overline{w}', \overline{q}')$ in $db(R)$ such that $\overline{w}' \neq \overline{w}$. From this, it follows that also $[\phi]^{db}$ holds.

We now prove our claim that $[\phi]^{db'}$ holds. Assume, for contradiction's sake, that this is not the case. This means that there is no tuple $(\overline{v}', \overline{w}', \overline{q}')$ in $db'(R)$ such that $\overline{v}' = \overline{v}$ and $\overline{w}' \neq \overline{w}$. Therefore, for all tuples $(\overline{v}', \overline{w}', \overline{q}')$ in $db'(R)$, if $\overline{v} = \overline{v}'$, then $\overline{w}' = \overline{w}$. From this and $db'[R \oplus (\overline{v}, \overline{w}, \overline{q})](R) = db'(R) \cup \{(\overline{v}, \overline{w}, \overline{q})\}$, it follows that for all tuples $(\overline{v}', \overline{w}', \overline{q}')$ in $db'[R \oplus (\overline{v}, \overline{w}, \overline{q})](R)$, if $\overline{v} = \overline{v}'$, then $\overline{w}' = \overline{w}$. Furthermore, from $db' \in \Omega_D^\Gamma$, it follows that for all tuples $(\overline{v}', \overline{w}', \overline{q}')$ and $(\overline{v}', \overline{w}'', \overline{q}'')$ in $db(R)$ such that $\overline{v}' \neq \overline{v}$, then $\overline{w}' = \overline{w}$. From this and $db[R \oplus (\overline{v}, \overline{w}, \overline{q})](R) = db'(R) \cup \{(\overline{v}, \overline{w}, \overline{q})\}$, it follows that for all tuples $(\overline{v}', \overline{w}', \overline{q}')$ and $(\overline{v}', \overline{w}'', \overline{q}'')$ in $db'[R \oplus (\overline{v}, \overline{w}, \overline{q})](R)$ such that $\overline{v}' \neq \overline{v}$, then $\overline{w}' = \overline{w}$. From these facts and the relational calculus semantics, it follows that $[\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. ((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}']^{db'[R \oplus (\overline{v}, \overline{w}, \overline{q})]} = \top$. This is in contradiction with the fact that the constraint $\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. ((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}'$ is in $Ex(last(r^i))$.

11. *`INSERT` ID Exception.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}) \rangle \cdot s$, where $s = \langle db, U, sec, T, V, c \rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c' \rangle$, and ϕ be $\forall \overline{x}, \overline{y}. S(\overline{x}, \overline{y}) \Rightarrow \overline{x} \neq \overline{v}$. We claim that $[\phi]^{db'}$ holds. From this claim and the LTS semantics, it follows that there is no tuple $(\overline{v}, \overline{w}')$ in $db'(S)$. From this and $db(S) = db'(S)$, it follows that there no tuple $(\overline{v}, \overline{w}')$ in $db(S)$. From this, it follows that also $[\phi]^{db}$ holds.

    We now prove our claim that $[\phi]^{db'}$ holds. Assume, for contradiction's sake, that this is not the case. This means that there is a tuple $(\overline{v}, \overline{w}')$ in $db'(S)$, for some $\overline{w}'$. From $db' \in \Omega_D^\Gamma$, it follows that for all tuples $(\overline{x}, \overline{z}) \in db'(R)$ such that $\overline{x} \neq \overline{v}$, there is a tuple $(\overline{x}, \overline{y}) \in db'(S)$. From this, $(\overline{v}, \overline{w}')$ in $db'(S)$, $db'[R \oplus (\overline{v}, \overline{w})](S) = db'(S)$, and $db'[R \oplus (\overline{v}, \overline{w})](R) = db'(R) \cup \{(\overline{v}, \overline{w})\}$, it follows that for all tuples $(\overline{x}, \overline{z}) \in db'[R \oplus (\overline{v}, \overline{w})](R)$, there is a tuple $(\overline{x}, \overline{y}) \in db'[R \oplus (\overline{v}, \overline{w})](S)$. From these facts and the relational calculus semantics, it follows that $[\forall \overline{x}, \overline{z}. (R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w}))]^{db'[R \oplus (\overline{v}, \overline{w})]} = \top$. This is in contradiction with the fact that the constraint $\forall \overline{x}, \overline{z}. (R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w}))$ is in $Ex(last(r^i))$.

12. *`DELETE` FD Exception.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{DELETE}, R, (\overline{v}, \overline{w}) \rangle \cdot s$, where $s = \langle db, U, sec, T, V, c \rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c' \rangle$, and ϕ be $\exists \overline{z}. S(\overline{v}, \overline{z}) \wedge \forall \overline{y}. (R(\overline{v}, \overline{y}) \Rightarrow \overline{y} = \overline{w})$. We claim that $[\phi]^{db'}$ holds. From this claim and the LTS semantics, it follows that there is a tuple $(\overline{v}, \overline{z})$ in $db'(S)$ and all tuples $(\overline{v}, \overline{y}) \in db'(R)$ are such that $\overline{y} = \overline{w}$. From $(\overline{v}, \overline{z})$ in $db'(S)$ and $db(S) = db'(S)$, it follows that $(\overline{v}, \overline{z})$ in $db'(S)$. From the fact that all tuples $(\overline{v}, \overline{y}) \in db'(R)$ are such that $\overline{y} = \overline{w}$ and $db(R) = db'(R)\}$, it follows that all tuples $(\overline{v}, \overline{y}) \in db(R)$ are such that $\overline{y} = \overline{w}$. From $(\overline{v}, \overline{z})$ in $db(S)$ and the fact that all tuples $(\overline{v}, \overline{y}) \in db(R)$ are such that $\overline{y} = \overline{w}$, it follows that $[\phi]^{db}$ holds.

    We now prove our claim that $[\phi]^{db'}$ holds. Assume, for contradiction's sake, that this is not the case. There are two cases:

    (a) all tuples $(\overline{x}, \overline{y}) \in db'(S)$ are such that $\overline{v} \neq \overline{x}$. Furthermore, from $db' \in \Omega_D^\Gamma$, it follows that for all tuples $(\overline{x}, \overline{y}) \in db(S)$ such that $\overline{v} \neq \overline{x}$, there is a tuple $(\overline{x}, \overline{z}) \in db(R)$. From these facts, $db'[R \ominus (\overline{v}, \overline{w})](S) = db'(S)$, and $db'[R \ominus (\overline{v}, \overline{w})](R) = db'(R) \setminus \{(\overline{v}, \overline{w})\}$, it follows that for all tuples $(\overline{x}, \overline{y}) \in db'[R \ominus (\overline{v}, \overline{w})](S)$, there is a tuple $(\overline{x}, \overline{z}) \in db'[R \ominus (\overline{v}, \overline{w})](R)$. From this and the relational calculus semantics, it follows that

    $$[\forall \overline{x}, \overline{z}. (S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. R(\overline{x}, \overline{w})]^{db'[R \ominus (\overline{v}, \overline{w}))]} = \top.$$

    This contradicts the fact that the constraint $\forall \overline{x}, \overline{z}. (S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. R(\overline{x}, \overline{w}))$ is in $Ex(last(r^i))$.

    (b) there is a tuple $(\overline{v}, \overline{w}') \in db'(R)$ such that $\overline{w} \neq \overline{w}'$. Furthermore, from $db' \in \Omega_D^\Gamma$, it follows that for all tuples $(\overline{x}, \overline{y}) \in db'(S)$ such that $\overline{v} \neq \overline{x}$, there is a tuple $(\overline{x}, \overline{z}) \in db'(R)$. From these facts, $db'[R \ominus (\overline{v}, \overline{w})](S) = db'(S)$, and $db'[R \ominus (\overline{v}, \overline{w})](R) = db'(R) \setminus \{(\overline{v}, \overline{w})\}$, it follows that for all tuples $(\overline{x}, \overline{y}) \in db'[R \ominus (\overline{v}, \overline{w})](S)$, there is a tuple $(\overline{x}, \overline{z}) \in db'[R \ominus (\overline{v}, \overline{w})](R)$. From this and the relational calculus semantics, it follows that

    $$[\forall \overline{x}, \overline{z}. (S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. R(\overline{x}, \overline{w})]^{db'[R \ominus (\overline{v}, \overline{w}))]} = \top.$$

    This contradicts the fact that the constraint $\forall \overline{x}, \overline{z}. (S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. R(\overline{x}, \overline{w}))$ is in $Ex(last(r^i))$.

13. *Integrity Constraint.* The proof of this case follows trivially from the fact that for any state $s = \langle db, U, sec, T, V, c \rangle \in \Omega_M$ and any $\gamma \in \Gamma$, $[\gamma]^{db} = \top$ holds by definition.

14. *Learn `GRANT/REVOKE` Backward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c \rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec', T, V, c' \rangle$, and t be a trigger whose `WHEN` condition is ϕ and whose action is either a `GRANT` or a `REVOKE`. From the rule's definition, it follows $sec \neq sec'$. We now prove that $[\phi]^{db} = \top$. Assume, for contradiction's sake, that $[\phi]^{db} = \bot$. From this and the LTS rules for the triggers, it follows that the trigger t is disabled. Therefore, according to the *Trigger Disabled* rule, $sec = sec'$, which leads to a contradiction.

15. *Trigger `GRANT` Disabled Backward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c \rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec', T, V, c' \rangle$, and t be a trigger whose `WHEN` condition is ψ, and ϕ be $\neg\psi$. Furthermore, let $g \in \Omega_{\mathcal{U},D}^{sec}$ be the `GRANT` added by the trigger. From the rule's definition, it follows $g \notin sec'$. We now prove that $[\phi]^{db} = \top$. Assume, for contradiction's sake, that $[\phi]^{db} = \bot$. This would imply that the trigger t is enabled. There are two cases:

    (a) t's execution is authorized. Therefore, $g \in sec'$, which contradicts $g \notin sec'$.

    (b) t's execution is not authorized. This contradicts $secEx(s) = \bot$.

16. *Trigger `REVOKE` Disabled Backward.* The proof for this case is similar to that of *Trigger `GRANT` Disabled Backward.*

17. *Trigger `INSERT` FD Exception.* The proof for this case is similar to that of *`INSERT` FD Exception.*

18. *Trigger `INSERT` ID Exception.* The proof for this case is similar to that of *`INSERT` ID Exception.*

19. *Trigger `DELETE` ID Exception.* The proof for this case is similar to that of *`DELETE` ID Exception.*

20. *Trigger Exception.* Let i be such that $r^i = r^{i-1} \cdot t \cdot$

s, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec', T, V, c'\rangle$, and t be a trigger whose `WHEN` condition is ϕ and whose action is act. From the rule's definition, it follows that t is enabled and that the evaluation of the `WHEN` condition is authorized. From this and the LTS's rules, it follows that $[\phi]^{db} = \top$.
21. *Trigger `INSERT` Exception.* The proof for this case is similar to that of *`INSERT` Exception.*
22. *Trigger `DELETE` Exception.* The proof for this case is similar to that of *`DELETE` Exception.*
23. *Trigger Rollback `INSERT`.* Let i be such that $r^i = r^{i-n-1} \cdot \langle u, \texttt{INSERT}, R, \bar{t}\rangle \cdot s_1 \cdot t_1 \cdot s_2 \cdot \ldots \cdot t_n \cdot s_n$, where $s_1, s_2, \ldots, s_n \in \Omega_M$ and $t_1, \ldots, t_n \in \mathcal{TRIGGER}_D$, and ϕ be $\neg R(\bar{t})$. Furthermore, let $last(r^{i-n-1}) = \langle db', U', sec', T', V', c'\rangle$ and s_n be $\langle db, U, sec, T, V, c\rangle$. Assume, for contradiction's sake, that $[\phi]^{db} = \bot$. Therefore, $\bar{t} \in db(R)$. From the LTS rules, it follows that $db' = db$. From this and $\bar{t} \in db(R)$, it follows $\bar{t} \in db'(R)$. From r's definition and the LTS rule *`INSERT` Success - 2*, it follows that $\bar{t} \notin db'(R)$, which leads to a contradiction.
24. *Trigger Rollback `DELETE`.* The proof for this case is similar to that of *Trigger Rollback `INSERT`.*

This completes the proof of the base step.

Induction Step: Assume that the claim hold for any derivation of $r, j \vdash_u \psi$ such that $|r, j \vdash_u \psi| < |r, i \vdash_u \phi|$. We now prove that the claim also holds for $r, i \vdash_u \phi$. There are a number of cases depending on the rule used to obtain $r, i \vdash_u \phi$.

1. *View.* The proof of this case follows trivially from the semantics of the relational calculus extended over views.
2. *Propagate Forward `SELECT`.* Let i be such that $r^{i+1} = r^i \cdot \langle u, \texttt{SELECT}, \psi\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^i) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i \vdash_u \phi$ holds. From this, the induction hypothesis, and $last(r^i) = \langle db', U', sec', T', V', c'\rangle$, it follows that $[\phi]^{db'} = \top$. From the LTS semantics, it follows that $db = db'$. From this and $[\phi]^{db'} = \top$, it follows that $[\phi]^{db} = \top$.
3. *Propagate Forward `GRANT/REVOKE`.* The proof for this case is similar to that of *Propagate Forward `SELECT`.*
4. *Propagate Forward `CREATE`.* The proof for this case is similar to that of *Propagate Forward `SELECT`.*
5. *Propagate Backward `SELECT`.* Let i be such that $r^{i+1} = r^i \cdot \langle u, \texttt{SELECT}, \psi\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$ and $last(r^i) = \langle db, U, sec, T, V, c\rangle$. From the rule's definition, $r, i+1 \vdash_u \phi$ holds. From this, the induction hypothesis, $r^{i+1} = r^i \cdot \langle u, \texttt{SELECT}, \psi\rangle \cdot s$, and $s = \langle db, U, sec, T, V, c\rangle$, it follows that $[\phi]^{db'} = \top$. From the LTS semantics, it follows that $db = db'$. From this and $[\phi]^{db'} = \top$, it follows that $[\phi]^{db} = \top$.
6. *Propagate Backward `GRANT/REVOKE`.* The proof for this case is similar to that of *Propagate Backward `SELECT`.*
7. *Propagate Backward `CREATE TRIGGER`.* The proof for this case is similar to that of *Propagate Backward `SELECT`.*
8. *Propagate Backward `CREATE VIEW`.* Let i be such that $r^{i+1} = r^i \cdot \langle u, \texttt{CREATE}, o\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$ and $last(r^i) = \langle db, U, sec, T, V, c\rangle$. From the rule's definition, $r, i+1 \vdash_u \phi'$ holds. From this, the induction hypothesis, $r^{i+1} = r^i \cdot \langle u, \texttt{SELECT}, \psi\rangle \cdot s$, and $s = \langle db, U, sec, T, V, c\rangle$, it follows that $[\phi']^{db'} = \top$. From the definition of *replace*, it follows that $replace(\phi', o)$ and ϕ' are semantically equivalent. From this and $[\phi']^{db'} = \top$, $[replace(\phi', o)]^{db'} = \top$. From the LTS semantics, it follows that $db = db'$. From this and $[replace(\phi', o)]^{db'} = \top$, it follows that $[replace(\phi', o)]^{db} = \top$.
9. *Rollback Backward - 1.* Let i be such that $r^i = r^{i-n-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s_1 \cdot t_1 \cdot s_2 \cdot \ldots \cdot t_n \cdot s_n$, where $s_1, s_2, \ldots, s_n \in \Omega_M$, $t_1, \ldots, t_n \in \mathcal{TRIGGER}_D$, and op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$. Furthermore, let s_n be $\langle db', U', sec', T', V', c'\rangle$ and $last(r^{i-n-1})$ be $\langle db, U, sec, T, V, c\rangle$. From the rule's definition, $r, i \vdash_u \phi$ holds. From this, the induction hypothesis, and $s_n = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, it follows that $[\phi]^{db'} = \top$. From the LTS semantics, it follows that $db = db'$ (because a roll-back happened). From this and $[\phi]^{db'} = \top$, it follows that $[\phi]^{db} = \top$.
10. *Rollback Backward - 2.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$. From the rule's definition, $r, i \vdash_u \phi$ holds. From this, the induction hypothesis, $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, and $s = \langle db', U', sec', T', V', c'\rangle$, it follows that $[\phi]^{db'} = \top$. From the LTS semantics, it follows that $db = db'$ (because a roll-back happened). From this and $[\phi]^{db'} = \top$, it follows that $[\phi]^{db} = \top$.
11. *Rollback Forward - 1.* Let i be such that $r^i = r^{i-n-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s_1 \cdot t_1 \cdot s_2 \cdot \ldots \cdot t_n \cdot s_n$, where $s_1, s_2, \ldots, s_n \in \Omega_M$, $t_1, \ldots, t_n \in \mathcal{TRIGGER}_D$, and op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$. Furthermore, let s_n be $\langle db, U, sec, T, V, c\rangle$ and $last(r^{i-n-1})$ be $\langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-n-1 \vdash_u \phi$ holds. From this, the induction hypothesis, and $last(r^{i-n-1}) = \langle db', U', sec', T', V', c'\rangle$., it follows that $[\phi]^{db'} = \top$. From the LTS semantics, it follows that $db = db'$ (because a roll-back happened). From this and $[\phi]^{db'} = \top$, it follows that $[\phi]^{db} = \top$.
12. *Rollback Forward - 2.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, where $op \in \{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this, the induction hypothesis, and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$, it follows that $[\phi]^{db'} = \top$. From the LTS semantics, it follows that $db = db'$ (because a roll-back happened). From this and $[\phi]^{db'} = \top$, it follows that $[\phi]^{db} = \top$.
13. *Propagate Forward `INSERT/DELETE` Success.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, where $op \in \{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this, the induction hypothesis, and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$, it follows that $[\phi]^{db'} = \top$. From $reviseBelief(r^{i-1}, \phi, r^i) = \top$, it follows that R does not occur in ϕ. From the LTS semantics, it follows that $db(R') = db'(R')$ for all $R' \neq R$. From this and the fact that R does not occur in ϕ, it follows $[\phi]^{db} = \top$.
14. *Propagate Forward `INSERT` Success - 1.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, where op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ and $r, i-1 \vdash_u R(\bar{t})$ hold.

From this, the induction hypothesis, and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$, it follows that $[\phi]^{db'} = \top$ and $[R(\bar{t})]^{db'} = \top$. From $[R(\bar{t})]^{db'} = \top$ and the relational calculus' semantics, it follows that $\bar{t} \in db'(R)$. From the LTS semantics, $db = db'[R \oplus \bar{t}]$. From this, it follows that $db(R') = db'(R')$ for all $R' \neq R$ and $db(R) = db'(R) \cup \{\bar{t}\}$. From this and $\bar{t} \in db'(R)$, it follows that $db(R) = db'(R)$. Therefore, $db = db'$. From this and $[\phi]^{db'} = \top$, it follows that $[\phi]^{db} = \top$.

15. *Propagate Forward* `DELETE` *Success - 1.* The proof for this case is similar to that of *Propagate Forward* `INSERT` *Success - 1.*
16. *Propagate Backward* `INSERT/DELETE` *Success.* The proof for this case is similar to that of *Propagate Forward* `INSERT/DELETE` *Success.*
17. *Propagate Backward* `INSERT` *Success - 1.* The proof for this case is similar to that of *Propagate Forward* `INSERT` *Success - 1.*
18. *Propagate Backward* `DELETE` *Success - 1.* The proof for this case is similar to that of *Propagate Forward* `DELETE` *Success - 1.*
19. *Reasoning.* Let Φ be a subset of $\{\phi \mid r, i \vdash_u \phi\}$ and $last(r^i) = \langle db, U, sec, T, V, c\rangle$. From the induction hypothesis, it follows that $[\phi]^{db} = \top$ for any $\phi \in \Phi$. From the rule's definition, it follows that $\Phi \models_{fin} \gamma$. From this and $[\phi]^{db} = \top$ for any $\phi \in \Phi$, it follows that $[\gamma]^{db} = \top$.
20. *Learn* `INSERT` *Backward - 3.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \bar{t}\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and ϕ be $\neg R(\bar{t})$. We prove that $[\neg R(\bar{t})]^{db} = \top$. Assume, for contradiction's sake, that $[\neg R(\bar{t})]^{db} = \bot$. From this and the relational calculus semantics, it follows that $\bar{t} \in db(R)$. From this and the LTS semantics, it follows that $db = db'$ because $db' = db[R \oplus \bar{t}]$. However, from the rule's definition, there is a ψ such that $r, i-1 \vdash_u \psi$ and $r, i \vdash_u \neg\psi$ hold. From this, the induction hypothesis, $s = \langle db', U', sec', T', V', c'\rangle$, and $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, it follows that $[\psi]^{db} = \top$ and $[\neg\psi]^{db'} = \top$. Therefore, $[\psi]^{db} = \top$ and $[\psi]^{db'} = \bot$. Hence, $db \neq db'$ leading to a contradiction with $db = db'$.
21. *Learn* `DELETE` *Backward - 3.* The proof for this case is similar to that of *Learn* `INSERT` *Backward - 3.*
22. *Propagate Forward Disabled Trigger.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and t be a trigger. Furthermore, let ψ be t's condition where all free variables are replaced with the values in $tpl(last(r^{i-1}))$. From the rule's definition, it follows that $r, i-1 \vdash_u \neg\psi$ holds. From this and the induction hypothesis, it follows that $[\psi]^{db'} = \bot$. From this, the fact that ψ is t's `WHEN` condition, and the rule *Trigger Disabled*, it follows that $db = db'$. From the rule's definition, it follows that $r, i-1 \vdash_u \phi$ holds. From this, the induction hypothesis, and $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, it follows that $[\phi]^{db'} = \top$. From this and $db = db'$, it follows that $[\phi]^{db} = \top$.
23. *Propagate Backward Disabled Trigger.* The proof for this case is similar to that of *Propagate Forward Disabled Trigger.*
24. *Learn* `INSERT` *Forward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and t be a trigger, and ϕ be $R(\bar{t})$. Furthermore, let ψ be t's condition where all free variables are replaced with the values in $tpl(last(r^{i-1}))$. From the rule's definition, it follows that $r, i-1 \vdash_u \psi$ holds. From this and the induction hypothesis, it follows that $[\psi]^{db'} = \bot$. Furthermore, from the rule's definition, it follows that $secEx(s) = \bot$ and $Ex(s) = \emptyset$. From this, the fact that ψ is t's `WHEN` condition, $[\psi]^{db'} = \bot$, and the rule *Trigger* `DELETE-INSERT` *Success*, it follows that $db = db'[R \oplus \bar{t}]$. From the definition of $\oplus$, it follows that $\bar{t} \in db(R)$. From this and the relational calculus semantics, it follows that $[\phi]^{db} = \top$.
25. *Learn* `INSERT` *- FD.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$, and $t \in \mathcal{TRIGGER}_D$, and ϕ be $\neg\exists \overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. Furthermore, let ψ be t's condition where all free variables are replaced with the values in $tpl(last(r^{i-1}))$ and $\langle u', \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle$ be t's actual action. We claim that $db(R) = db'(R) \cup \{(\overline{v}, \overline{w}, \overline{q})\}$. Furthermore, we claim that $[\phi]^{db}$ holds. From this claim and the relational calculus semantics, it follows that there is no tuple $(\overline{v}', \overline{w}', \overline{q}')$ in $db(R)$ such that $\overline{v}' = \overline{v}$ and $\overline{w}' \neq \overline{w}$. From this and $db(R) = db'(R) \cup \{(\overline{v}, \overline{w}, \overline{q})\}$, it follows that there is no tuple $(\overline{v}', \overline{w}', \overline{q}')$ in $db'(R)$ such that $\overline{v}' = \overline{v}$ and $\overline{w}' \neq \overline{w}$. From this, it follows that also $[\phi]^{db'}$ holds.
We now prove our claim that $db(R) = db'(R) \cup \{(\overline{v}, \overline{w}, \overline{q})\}$. Assume, for contradiction's sake, that this is not the case. Since db is obtained from db', this would imply that the trigger t is disabled. Hence, this would imply that $[\psi]^{db'} = \bot$. From the rule's definition, $r, i-1 \vdash_u \psi$. From this, the induction's hypothesis, and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, it follows that $[\psi]^{db'} = \top$, which contradicts $[\psi]^{db'} = \bot$.
We now prove our claim that $[\phi]^{db}$ holds. Assume, for contradiction's sake, that this is not the case. This means that there is a tuple $(\overline{v}', \overline{w}', \overline{q}')$ in $db(R)$ such that $\overline{v}' = \overline{v}$ and $\overline{w}' \neq \overline{w}$. Note that, as we proved before, $(\overline{v}, \overline{w}, \overline{q}) \in db(R)$. Therefore, there are two tuples $(\overline{v}, \overline{w}, \overline{q})$ and $(\overline{v}, \overline{w}', \overline{q}')$ in $db(R)$ such that $\overline{w}' \neq \overline{w}$. From this and the relational calculus semantics, it follows that $[\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'.\, ((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}']^{db} = \bot$. This is in contradiction with the fact that the constraint $\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'.\, ((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}'$ is in Γ. Indeed, since the constraint is in Γ, any state in Ω_D^Γ must satisfy it.
26. *Learn* `INSERT` *- FD - 1.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$, and $t \in \mathcal{TRIGGER}_D$, and ϕ be $\neg\exists \overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. Furthermore, let ψ be t's condition where all free variables are replaced with the values in $tpl(last(r^{i-1}))$ and $\langle u', \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle$ be t's actual action. From the rule's definition, $r, i-1 \vdash_u \psi$. From this, the induction's hypothesis, and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, it follows that $[\psi]^{db'} = \top$. From this and the LTS semantics, it follows that the trigger t is enabled in $last(r^{i-1})$. We now prove our claim that $[\phi]^{db'}$ holds. Assume, for contradiction's sake, that this is not the case. This means that there is a tuple $(\overline{v}', \overline{w}', \overline{q}')$ in $db'(R)$ such that $\overline{v}' = \overline{v}$ and $\overline{w}' \neq \overline{w}$. Let db'' be the state $db'[R \oplus (\overline{v}, \overline{w}, \overline{q})]$. From $db'' = db'[R \oplus (\overline{v}, \overline{w}, \overline{q})]$, and the fact that there is a tuple $(\overline{v}', \overline{w}', \overline{q}')$

in $db'(R)$ such that $\overline{v}' = \overline{v}$ and $\overline{w}' \neq \overline{w}$, it follows that there are two tuples $(\overline{v},\overline{w},\overline{q})$ and $(\overline{v},\overline{w}',\overline{q}')$ in $db''(R)$ such that $\overline{w}' \neq \overline{w}$. From this and the relational calculus semantics, it follows that $[\forall \overline{x},\overline{y},\overline{y}',\overline{z},\overline{z}'.\,((R(\overline{x},\overline{y},\overline{z}) \wedge R(\overline{x},\overline{y}',\overline{z}')) \Rightarrow \overline{y} = \overline{y}']^{db''} = \bot$. Since the trigger t is enabled, this contradicts the fact that $\forall \overline{x},\overline{y},\overline{y}',\overline{z},\overline{z}'.\,((R(\overline{x},\overline{y},\overline{z}) \wedge R(\overline{x},\overline{y}',\overline{z}')) \Rightarrow \overline{y} = \overline{y}'$ is not in $Ex(s)$.

27. *Learn `INSERT` - ID.* The proof of this case is similar to that of *Learn `INSERT` - FD*. See also the proof of *`INSERT` Success - ID*.
28. *Learn `INSERT` - ID - 1.* The proof of this case is similar to that of *Learn `INSERT` - FD - 1*. See also the proof of *`INSERT` Success - ID*.
29. *Learn `INSERT` Backward - 1.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db', U', sec', T', V', c' \rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c \rangle$, and $t \in \mathcal{TRIGGER}_D$, and ϕ be t's actual `WHEN` condition, where all free variables are replaced with the values in $tpl(last(r^{i-1}))$. From the rule's definition, it follows that there is a ψ such that $r, i-1 \vdash_u \psi$ and $r, i \vdash_u \neg\psi$. From this, the induction's hypothesis, $s = \langle db', U', sec', T', V', c' \rangle$, and $last(r^{i-1}) = \langle db, U, sec, T, V, c \rangle$, it follows that $[\psi]^{db} = \top$ and $[\neg\psi]^{db'} = \top$. Therefore, $[\psi]^{db} = \top$ and $[\psi]^{db'} = \bot$. Hence, $db \neq db'$. We now prove that $[\phi]^{db} = \top$. Assume, for contradiction's sake, that $[\phi]^{db} = \bot$. From the rule's definition, it follows that $secEx(s) = \bot$. Therefore, $f(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi \rangle) = \top$. From this, $[\phi]^{db} = \bot$, and the rule `Trigger Disabled`, it follows that $db = db'$, which contradicts $db \neq db'$.
30. *Learn `INSERT` Backward - 2.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db', U', sec', T', V', c' \rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c \rangle$, and $t \in \mathcal{TRIGGER}_D$, and ϕ be $\neg R(\overline{t})$. Furthermore, let $act = \langle u', \texttt{INSERT}, R, \overline{t} \rangle$ be t's actual action. From the rule's definition, it follows that there is a ψ such that $r, i-1 \vdash_u \psi$ and $r, i \vdash_u \neg\psi$. From this, the induction's hypothesis, $s = \langle db', U', sec', T', V', c' \rangle$, and $last(r^{i-1}) = \langle db, U, sec, T, V, c \rangle$, it follows that $[\psi]^{db} = \top$ and $[\neg\psi]^{db'} = \top$. Therefore, $[\psi]^{db} = \top$ and $[\psi]^{db'} = \bot$. Hence, $db \neq db'$. We now prove that $[\phi]^{db} = \top$. Assume, for contradiction's sake, that $[\phi]^{db} = \bot$. Therefore, $\overline{t} \in db(R)$. From this and $act = \langle u', \texttt{INSERT}, R, \overline{t} \rangle$, it follows that $db' = db[R \oplus \overline{t}]$. From this and $\oplus$'s definition, it follows that $db'(R') = db(R')$ for all $R' \neq R$ and $db'(R) = db(R) \cup \{\overline{t}\}$. From $db'(R) = db(R) \cup \{\overline{t}\}$ and $\overline{t} \in db(R)$, it follows that $db'(R) = db(R)$. From this and $db'(R') = db(R')$ for all $R' \neq R$, it follows that $db' = db$, which contradicts $db \neq db'$.
31. *Learn `DELETE` Forward.* The proof of this case is similar to that of *Learn `INSERT` Forward*.
32. *Learn `DELETE` - ID.* The proof of this case is similar to that of *Learn `INSERT` - FD*. See also the proof of *`DELETE` Success - ID*.
33. *Learn `DELETE` - ID - 1.* The proof of this case is similar to that of *Learn `INSERT` - FD - 1*. See also the proof of *`DELETE` Success - ID*.
34. *Learn `DELETE` Backward - 1.* The proof of this case is similar to that of *Learn `INSERT` Backward - 1*.
35. *Learn `DELETE` Backward - 2.* The proof of this case is similar to that of *Learn `INSERT` Backward - 2*.
36. *Propagate Forward Trigger Action.* The proof of this case is similar to *Propagate Forward `INSERT/DELETE` Success*.
37. *Propagate Backward Trigger Action.* The proof of this case is similar to *Propagate Backward `INSERT/DELETE` Success*.
38. *Propagate Forward `INSERT` Trigger Action.* The proof of this case is similar to that of *Propagate Forward `INSERT` Success - 1*.
39. *Propagate Forward `DELETE` Trigger Action.* The proof of this case is similar to that of *Propagate Forward `DELETE` Success - 1*.
40. *Propagate Backward `INSERT` Trigger Action.* The proof of this case is similar to that of *Propagate Backward `INSERT` Success - 1*.
41. *Propagate Backward `DELETE` Trigger Action.* The proof of this case is similar to that of *Propagate Backward `DELETE` Success - 1*.
42. *Trigger FD `INSERT` Disabled Backward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db', U', sec', T', V', c' \rangle \in \Omega_M$, $t \in \mathcal{TRIGGER}_D$, and $last(r^{i-1}) = \langle db, U, sec, T, V, c \rangle$, and ψ be $\neg\phi[\overline{x}^{|R'|} \mapsto tpl(last(r^{i-1}))]$. Furthermore, let $act = \langle u', \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q}) \rangle$ be t's actual action. From the rule's definition, it follows that $r, i-1 \vdash_u \exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$ holds. From this, the induction hypothesis, and $last(r^{i-1}) = \langle db, U, sec, T, V, c \rangle$, it follows that $[\exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}]^{db} = \top$. Therefore, there is a tuple $(\overline{v}, \overline{w}', \overline{z}') \in db(R)$ such that $\overline{w}' \neq \overline{w}$. We now prove that $[\psi]^{db} = \top$. Assume, for contradiction's sake, that this is not the case, namely that $[\phi[\overline{x} \mapsto tpl(last(r^{i-1}))]]^{db} = \top$. There are two cases:
    (a) the trigger t is enabled and the action act is authorized. In this case, the database $db[R \oplus \{(\overline{v}, \overline{w}, \overline{q})\}] \notin \Omega_D^\Gamma$ because $\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'.\,(R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}' \in \Gamma$ and there is a tuple $(\overline{v}, \overline{w}', \overline{z}') \in db(R)$ such that $\overline{w}' \neq \overline{w}$. Therefore, the resulting state would be such that $Ex(s) \neq \emptyset$. This contradicts the fact that, according to the rule's definition, $Ex(s) = \emptyset$.
    (b) the trigger t is enabled and the action act is not authorized. Therefore, the resulting state would be such that $secEx(s) = \top$. This contradicts the fact that, according to the rule's definition, $secEx(s) = \bot$.
43. *Trigger ID `INSERT` Disabled Backward.* The proof of this case is similar to that of *Trigger FD `INSERT` Disabled Backward*.
44. *Trigger ID `DELETE` Disabled Backward.* The proof of this case is similar to that of *Trigger FD `INSERT` Disabled Backward*.

This completes the proof of the induction step.

This completes the proof of the theorem. $\square$

$$\frac{r, i \vdash_u \psi \quad r^{i+1} = r^i \cdot \langle u, \texttt{SELECT}, \phi\rangle \cdot s \quad 1 \leq i < |r| \quad s \in \Omega_M}{r, i+1 \vdash_u \psi} \text{ Propagate Forward } \texttt{SELECT}$$

$$\frac{r, i \vdash_u \psi \quad r^{i+1} = r^i \cdot \langle op, u', pr, u\rangle \cdot s \quad 1 \leq i < |r| \quad op \in \{\oplus, \oplus^*, \ominus\} \quad s \in \Omega_M}{r, i+1 \vdash_u \psi} \text{ Propagate Forward } \texttt{GRANT/REVOKE}$$

$$\frac{r, i \vdash_u \psi \quad r^{i+1} = r^i \cdot \langle u, \texttt{CREATE}, o\rangle \cdot s \quad 1 \leq i < |r| \quad o \in \mathcal{TRIGGER}_D \cup \mathcal{VIEW}_D \quad s \in \Omega_M}{r, i+1 \vdash_u \psi} \text{ Propagate Forward } \texttt{CREATE}$$

Figure 21: Rules defining how the attacker propagates (forward) the knowledge

$$\frac{r, i+1 \vdash_u \psi \quad r^{i+1} = r^i \cdot \langle u, \texttt{SELECT}, \phi\rangle \cdot s \quad 1 \leq i < |r| \quad s \in \Omega_M}{r, i \vdash_u \psi} \text{ Propagate Backward } \texttt{SELECT}$$

$$\frac{r, i+1 \vdash_u \psi \quad r^{i+1} = r^i \cdot \langle op, u', pr, u\rangle \cdot s \quad 1 \leq i < |r| \quad op \in \{\oplus, \oplus^*, \ominus\} \quad s \in \Omega_M}{r, i \vdash_u \psi} \text{ Propagate Backward } \texttt{GRANT/REVOKE}$$

$$\frac{r, i+1 \vdash_u \psi \quad r^{i+1} = r^i \cdot \langle u, \texttt{CREATE}, o\rangle \cdot s \quad 1 \leq i < |r| \quad o \in \mathcal{TRIGGER}_D \quad s \in \Omega_M}{r, i \vdash_u \psi} \text{ Propagate Backward } \texttt{CREATE TRIGGER}$$

$$\frac{r, i+1 \vdash_u \psi \quad r^{i+1} = r^i \cdot \langle u, \texttt{CREATE}, o\rangle \cdot s \quad 1 \leq i < |r| \quad o \in \mathcal{VIEW}_D \quad s \in \Omega_M \quad \psi' = replace(\psi, o)}{r, i \vdash_u \psi'} \text{ Propagate Backward } \texttt{CREATE VIEW}$$

Figure 22: Rules defining how the attacker propagates (backward) the knowledge

$$\frac{1 < i \leq |r| \quad r^i = r^{i-1} \cdot \langle u, \texttt{SELECT}, \phi\rangle \cdot s \quad s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{SELECT}, \phi\rangle, \top, \top, \emptyset\rangle, \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}{r, i \vdash_u \phi}\ \texttt{SELECT}\text{ Success - 1}$$

$$\frac{1 < i \leq |r| \quad r^i = r^{i-1} \cdot \langle u, \texttt{SELECT}, \phi\rangle \cdot s \quad s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{SELECT}, \phi\rangle, \top, \bot, \emptyset\rangle, \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle}{r, i \vdash_u \neg\phi}\ \texttt{SELECT}\text{ Success - 2}$$

$$\frac{1 < i \leq |r| \quad r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s \quad s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{INSERT}, R, \overline{t}\rangle, \top, \top, \emptyset\rangle, \langle rS', \overline{t}, u, tr\rangle\rangle}{r, i \vdash_u R(\overline{t})}\ \texttt{INSERT}\text{ Success}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s \qquad l \in \{i, i-1\} \\ s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{INSERT}, R, \overline{t}\rangle, \top, \top, E\rangle, \langle rS', \overline{t}, u, tr\rangle\rangle \\ \forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. \left((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}' \in \Gamma \setminus E \quad \overline{t} = (\overline{v}, \overline{w}, \overline{q}) \right. \end{array}}{r, l \vdash_u \neg\exists \overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}}\ \texttt{INSERT}\text{ Success - FD}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s \qquad l \in \{i, i-1\} \\ s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{INSERT}, R, \overline{t}\rangle, \top, \top, E\rangle, \langle rS', \overline{t}, u, tr\rangle\rangle \\ \forall \overline{x}, \overline{z}.\, (R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{y}.\, S(\overline{x}, \overline{y})) \in \Gamma \setminus E \qquad \overline{t} = (\overline{v}, \overline{w}) \end{array}}{r, l \vdash_u \exists \overline{y}.\, S(\overline{v}, \overline{y})}\ \texttt{INSERT}\text{ Success - ID}$$

$$\frac{1 < i \leq |r| \quad r^i = r^{i-1} \cdot \langle u, \texttt{DELETE}, R, \overline{t}\rangle \cdot s \quad s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{DELETE}, R, \overline{t}\rangle, \top, \top, \emptyset\rangle, \langle rS', \overline{t}, u, tr\rangle\rangle}{r, i \vdash_u \neg R(\overline{t})}\ \texttt{DELETE}\text{ Success}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot \langle u, \texttt{DELETE}, R, \overline{t}\rangle \cdot s \qquad l \in \{i, i-1\} \\ s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{DELETE}, R, \overline{t}\rangle, \top, \top, E\rangle, \langle rS', \overline{t}, u, tr\rangle\rangle \\ \forall \overline{x}, \overline{z}.\, (S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{y}.\, R(\overline{x}, \overline{y})) \in \Gamma \setminus E \qquad \overline{t} = (\overline{v}, \overline{w}) \end{array}}{r, l \vdash_u \forall \overline{x}, \overline{z}.\, (S(\overline{x}, \overline{z}) \Rightarrow \overline{x} \neq \overline{v}) \vee \exists \overline{y}.\, (R(\overline{v}, \overline{y}) \wedge \overline{y} \neq \overline{w})}\ \texttt{DELETE}\text{ Success - ID}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s \qquad l \in \{i, i-1\} \\ s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{INSERT}, R, \overline{t}\rangle, \top, \bot, E\rangle, \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle \quad E \neq \emptyset \end{array}}{r, l \vdash_u \neg R(\overline{t})}\ \texttt{INSERT}\text{ Exception}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot \langle u, \texttt{DELETE}, R, \overline{t}\rangle \cdot s \qquad l \in \{i, i-1\} \\ s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{DELETE}, R, \overline{t}\rangle, \top, \bot, E\rangle, \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle \quad E \neq \emptyset \end{array}}{r, l \vdash_u R(\overline{t})}\ \texttt{DELETE}\text{ Exception}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s \qquad l \in \{i, i-1\} \\ s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{INSERT}, R, \overline{t}\rangle, \top, \bot, E\rangle, \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle \\ (\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'.\, ((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}') \in E \quad \overline{t} = (\overline{v}, \overline{w}, \overline{q}) \end{array}}{r, l \vdash_u \exists \overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}}\ \texttt{INSERT}\text{ FD Exception}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s \qquad l \in \{i, i-1\} \\ s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{INSERT}, R, \overline{t}\rangle, \top, \bot, E\rangle, \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle \\ \forall \overline{x}, \overline{z}.\, (R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{y}.\, S(\overline{x}, \overline{y})) \in E \qquad \overline{t} = (\overline{v}, \overline{w}) \end{array}}{r, l \vdash_u \forall \overline{x}, \overline{y}.\, S(\overline{x}, \overline{y}) \Rightarrow \overline{x} \neq \overline{v}}\ \texttt{INSERT}\text{ ID Exception}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot \langle u, \texttt{DELETE}, R, \overline{t}\rangle \cdot s \qquad l \in \{i, i-1\} \\ s = \langle db, U, sec, T, V, h, \langle\langle u, \texttt{DELETE}, R, \overline{t}\rangle, \top, \bot, E\rangle, \langle \epsilon, \epsilon, \epsilon, \epsilon\rangle\rangle \\ \forall \overline{x}, \overline{z}.\, (S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{y}.\, R(\overline{x}, \overline{y})) \in E \qquad \overline{t} = (\overline{v}, \overline{w}) \end{array}}{r, l \vdash_u \exists \overline{z}.\, S(\overline{v}, \overline{z}) \wedge \forall \overline{y}.\, (R(\overline{v}, \overline{y}) \Rightarrow \overline{y} = \overline{w})}\ \texttt{DELETE}\text{ ID Exception}$$

$$\frac{1 \leq i \leq |r| \quad \gamma \in \Gamma}{r, i \vdash_u \gamma}\ \text{Integrity Constraint}$$

$$\frac{1 \leq i \leq |r| \quad v \in \mathit{last}(r^i).V \quad r, i \vdash_u \psi \quad \psi' = \mathit{replace}(\psi, v)}{r, i \vdash_u \psi'}\ \text{View}$$

Figure 23: Rules defining how the attacker extracts knowledge from the run

$$\frac{\begin{array}{c} r,i\vdash_u \phi \quad n+1<i\leq|r| \quad s_1,s_2,\ldots,s_n\in\Omega_M \quad t_1,\ldots,t_n\in\mathcal{TRIGGER}_D \\ secEx(s_n)=\top\vee Ex(s_n)\neq\emptyset \quad r^i=r^{i-n-1}\cdot\langle u,op,R,\bar{t}\rangle\cdot s_1\cdot t_1\cdot s_2\cdot\ldots\cdot t_n\cdot s_n \\ s_n=\langle db,U,sec,T,V,h,\langle t_n,when,stmt\rangle,\langle\epsilon,\epsilon,\epsilon,\epsilon\rangle\rangle \quad op\in\{\mathtt{INSERT},\mathtt{DELETE}\}\end{array}}{r,i-n-1\vdash_u \phi}\ \text{Rollback Backward - 1}$$

$$\frac{\begin{array}{c} r,i\vdash_u \phi \quad 1<i\leq|r| \quad secEx(s)=\top\vee Ex(s)\neq\emptyset \quad op\in\{\mathtt{INSERT},\mathtt{DELETE}\} \\ r^i=r^{i-1}\cdot\langle u,op,R,\bar{t}\rangle\cdot s \quad s=\langle db,U,sec,T,V,h,\langle\langle u,\mathtt{op},R,\bar{t}\rangle,v,v',E\rangle,\langle\epsilon,\epsilon,\epsilon,\epsilon\rangle\rangle\end{array}}{r,i-1\vdash_u \phi}\ \text{Rollback Backward - 2}$$

$$\frac{\begin{array}{c} r,i-n-1\vdash_u \phi \quad n+1<i\leq|r| \quad s_1,s_2,\ldots,s_n\in\Omega_M \quad t_1,\ldots,t_n\in\mathcal{TRIGGER}_D \\ secEx(s_n)=\top\vee Ex(s_n)\neq\emptyset \quad r^i=r^{i-n-1}\cdot\langle u,op,R,\bar{t}\rangle\cdot s_1\cdot t_1\cdot s_2\cdot\ldots\cdot t_n\cdot s_n \\ s_n=\langle db,U,sec,T,V,h,\langle t_n,when,stmt\rangle,\langle\epsilon,\epsilon,\epsilon,\epsilon\rangle\rangle \quad op\in\{\mathtt{INSERT},\mathtt{DELETE}\}\end{array}}{r,i\vdash_u \phi}\ \text{Rollback Forward - 1}$$

$$\frac{\begin{array}{c} r,i-1\vdash_u \phi \quad 1<i\leq|r| \quad secEx(s)=\top\vee Ex(s)\neq\emptyset \quad op\in\{\mathtt{INSERT},\mathtt{DELETE}\} \\ r^i=r^{i-1}\cdot\langle u,op,R,\bar{t}\rangle\cdot s \quad s=\langle db,U,sec,T,V,h,\langle\langle u,\mathtt{op},R,\bar{t}\rangle,v,v',E\rangle,\langle\epsilon,\epsilon,\epsilon,\epsilon\rangle\rangle\end{array}}{r,i\vdash_u \phi}\ \text{Rollback Forward - 2}$$

Figure 24: Rules regulating how information propagates in case of rollbacks

$$\frac{\begin{array}{c} 1<i\leq|r| \quad r,i-1\vdash_u \phi \quad r^i=r^{i-1}\cdot\langle u,op,R,\bar{t}\rangle\cdot s \\ s\in\Omega_M \quad secEx(s_n)=\bot \quad Ex(s_n)=\emptyset \quad s=\langle db,U,sec,T,V,h,actEff,tr\rangle \\ reviseBelief(r^{i-1},\phi,r^i)=\top \quad op\in\{\mathtt{INSERT},\mathtt{DELETE}\}\end{array}}{r,i\vdash_u \phi}\ \begin{array}{c}\text{Propagate Forward}\\ \mathtt{INSERT/DELETE}\text{ Success}\end{array}$$

$$\frac{\begin{array}{c} 1<i\leq|r| \quad r,i-1\vdash_u \phi \quad r,i-1\vdash_u R(\bar{t}) \quad r^i=r^{i-1}\cdot\langle u,\mathtt{INSERT},R,\bar{t}\rangle\cdot s \\ s\in\Omega_M \quad secEx(s_n)=\bot \quad Ex(s_n)=\emptyset \quad s=\langle db,U,sec,T,V,h,actEff,tr\rangle\end{array}}{r,i\vdash_u \phi}\ \begin{array}{c}\text{Propagate Forward}\\ \mathtt{INSERT}\text{ Success - 1}\end{array}$$

$$\frac{\begin{array}{c} 1<i\leq|r| \quad r,i-1\vdash_u \phi \quad r,i-1\vdash_u \neg R(\bar{t}) \quad r^i=r^{i-1}\cdot\langle u,\mathtt{DELETE},R,\bar{t}\rangle\cdot s \\ s\in\Omega_M \quad secEx(s_n)=\bot \quad Ex(s_n)=\emptyset \quad s=\langle db,U,sec,T,V,h,actEff,tr\rangle\end{array}}{r,i\vdash_u \phi}\ \begin{array}{c}\text{Propagate Forward}\\ \mathtt{DELETE}\text{ Success - 1}\end{array}$$

$$\frac{\begin{array}{c} 1<i\leq|r| \quad r,i\vdash_u \phi \quad r^i=r^{i-1}\cdot\langle u,op,R,\bar{t}\rangle\cdot s \\ s\in\Omega_M \quad secEx(s_n)=\bot \quad Ex(s_n)=\emptyset \quad s=\langle db,U,sec,T,V,h,actEff,tr\rangle \\ reviseBelief(r^{i-1},\phi,r^i)=\top \quad op\in\{\mathtt{INSERT},\mathtt{DELETE}\}\end{array}}{r,i-1\vdash_u \phi}\ \begin{array}{c}\text{Propagate Backward}\\ \mathtt{INSERT/DELETE}\text{ Success}\end{array}$$

$$\frac{\begin{array}{c} 1<i\leq|r| \quad r,i\vdash_u \phi \quad r,i-1\vdash_u R(\bar{t}) \quad r^i=r^{i-1}\cdot\langle u,\mathtt{INSERT},R,\bar{t}\rangle\cdot s \\ s\in\Omega_M \quad secEx(s_n)=\bot \quad Ex(s_n)=\emptyset \quad s=\langle db,U,sec,T,V,h,actEff,tr\rangle\end{array}}{r,i-1\vdash_u \phi}\ \begin{array}{c}\text{Propagate Backward}\\ \mathtt{INSERT}\text{ Success - 1}\end{array}$$

$$\frac{\begin{array}{c} 1<i\leq|r| \quad r,i\vdash_u \phi \quad r,i-1\vdash_u \neg R(\bar{t}) \quad r^i=r^{i-1}\cdot\langle u,\mathtt{DELETE},R,\bar{t}\rangle\cdot s \\ s\in\Omega_M \quad secEx(s_n)=\bot \quad Ex(s_n)=\emptyset \quad s=\langle db,U,sec,T,V,h,actEff,tr\rangle\end{array}}{r,i-1\vdash_u \phi}\ \begin{array}{c}\text{Propagate Backward}\\ \mathtt{DELETE}\text{ Success - 1}\end{array}$$

Figure 25: Rules regulating how information propagates in case of successful `INSERT` and `DELETE`

$$\frac{1\leq i\leq|r| \quad \Phi\subseteq\{\phi\,|\,r,i\vdash_u \phi\} \quad \Phi\models_{fin}\gamma}{r,i\vdash_u \gamma}\ \text{Reasoning}$$

Figure 26: Rules regulating the reasoning

$$\frac{\begin{array}{c} r^i=r^{i-1}\cdot\langle u,\mathtt{INSERT},R,\bar{t}\rangle\cdot s \quad 1<i\leq|r| \\ s=\langle db,U,sec,T,V,h,aE,tr\rangle \quad secEx(s)=\bot \\ Ex(s)=\emptyset \quad r,i-1\vdash_u \psi \quad r,i\vdash_u \neg\psi\end{array}}{r,i-1\vdash_u \neg R(\bar{t})}\ \text{Learn }\mathtt{INSERT}\text{ Backward - 3}$$

$$\frac{\begin{array}{c} r^i=r^{i-1}\cdot\langle u,\mathtt{DELETE},R,\bar{t}\rangle\cdot s \quad 1<i\leq|r| \\ s=\langle db,U,sec,T,V,h,aE,tr\rangle \quad secEx(s)=\bot \\ Ex(s)=\emptyset \quad r,i-1\vdash_u \psi \quad r,i\vdash_u \neg\psi\end{array}}{r,i-1\vdash_u R(\bar{t})}\ \text{Learn }\mathtt{DELETE}\text{ Backward - 3}$$

Figure 27: Rules describing how the attacker learns facts about `INSERT` and `DELETE` commands

$$\dfrac{\begin{array}{c} r, i-1 \vdash_u \phi \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, stmt \rangle, tr \rangle \qquad \mathit{secEx}(s) = \bot \\ t = \langle id, ow, ev, R, \psi, act, m \rangle \quad r, i-1 \vdash_u \neg\psi[\overline{x}^{|R|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))] \end{array}}{r, i \vdash_u \phi} \text{ Propagate Forward Disabled Trigger}$$

$$\dfrac{\begin{array}{c} r, i \vdash_u \phi \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, stmt \rangle, tr \rangle \qquad \mathit{secEx}(s) = \bot \\ t = \langle id, ow, ev, R, \psi, act, m \rangle \quad r, i-1 \vdash_u \neg\psi[\overline{x}^{|R|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))] \end{array}}{r, i-1 \vdash_u \phi} \text{ Propagate Backward Disabled Trigger}$$

Figure 28: Rules regulating the propagation of information through disabled triggers

$$\dfrac{\begin{array}{c} r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))] \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{INSERT}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad Ex(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \end{array}}{r, i \vdash_u R(\overline{t})} \text{ Learn } \texttt{INSERT} \text{ Forward}$$

$$\dfrac{\begin{array}{c} r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))] \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{INSERT}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \qquad l \in \{i, i-1\} \\ \mathit{secEx}(s) = \bot \qquad Ex(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\ \forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. ((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}' \in \Gamma \qquad \overline{t} = (\overline{v}, \overline{w}, \overline{q}) \end{array}}{r, l \vdash_u \neg \exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}} \text{ Learn } \texttt{INSERT} \text{ - FD}$$

$$\dfrac{\begin{array}{c} r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))] \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{INSERT}, R, \overline{t} \rangle, \top, \top, E \rangle \rangle, tr \rangle \quad \overline{t} = (\overline{v}, \overline{w}, \overline{q}) \quad \mathit{secEx}(s) = \bot \\ t = \langle id, ow, ev, R', \phi, act, m \rangle \qquad \forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. ((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}' \in \Gamma \setminus E \end{array}}{r, i-1 \vdash_u \neg \exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}} \text{ Learn } \texttt{INSERT} \text{ - FD - 1}$$

$$\dfrac{\begin{array}{c} r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))] \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{INSERT}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \qquad l \in \{i, i-1\} \\ \mathit{secEx}(s) = \bot \qquad Ex(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\ (\forall \overline{x}, \overline{z}. (R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w})) \in \Gamma \qquad \overline{t} = (\overline{v}, \overline{w}) \end{array}}{r, l \vdash_u \exists \overline{y}. S(\overline{v}, \overline{y})} \text{ Learn } \texttt{INSERT} \text{ - ID}$$

$$\dfrac{\begin{array}{c} r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))] \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{INSERT}, R, \overline{t} \rangle, \top, \top, E \rangle \rangle, tr \rangle \qquad \overline{t} = (\overline{v}, \overline{w}) \\ \mathit{secEx}(s) = \bot \quad t = \langle id, ow, ev, R', \phi, act, m \rangle \qquad (\forall \overline{x}, \overline{z}. (R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w})) \in \Gamma \setminus E \end{array}}{r, i-1 \vdash_u \exists \overline{y}. S(\overline{v}, \overline{y})} \text{ Learn } \texttt{INSERT} \text{ - ID - 1}$$

$$\dfrac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{INSERT}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad Ex(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\ r, i-1 \vdash_u \psi \qquad r, i \vdash_u \neg\psi \end{array}}{r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))]} \text{ Learn } \texttt{INSERT} \text{ Backward - 1}$$

$$\dfrac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{INSERT}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad Ex(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\ r, i-1 \vdash_u \psi \qquad r, i \vdash_u \neg\psi \end{array}}{r, i-1 \vdash_u \neg R(\overline{t})} \text{ Learn } \texttt{INSERT} \text{ Backward - 2}$$

Figure 29: Extracting knowledge from triggers

$$\frac{\begin{array}{c} r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))] \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{DELETE}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad Ex(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \end{array}}{r, i \vdash_u \neg R(\overline{t})} \text{ Learn } \texttt{DELETE} \text{ Forward}$$

$$\frac{\begin{array}{c} r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))] \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{DELETE}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \qquad l \in \{i, i-1\} \\ \mathit{secEx}(s) = \bot \qquad Ex(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\ (\forall \overline{x}, \overline{z}.\, (S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}.\, R(\overline{x}, \overline{w})) \in \Gamma \qquad \overline{t} = (\overline{v}, \overline{w}) \end{array}}{r, l \vdash_u \forall \overline{x}, \overline{z}.\, (S(\overline{x}, \overline{z}) \Rightarrow \overline{x} \neq \overline{v}) \vee \exists \overline{y}.\, (R(\overline{v}, \overline{y}) \wedge \overline{y} \neq \overline{w})} \text{ Learn } \texttt{DELETE} \text{ - ID}$$

$$\frac{\begin{array}{c} r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))] \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{DELETE}, R, \overline{t} \rangle, \top, \top, E \rangle \rangle, tr \rangle \qquad \overline{t} = (\overline{v}, \overline{w}) \\ \mathit{secEx}(s) = \bot \quad t = \langle id, ow, ev, R', \phi, act, m \rangle \quad (\forall \overline{x}, \overline{z}.\, (S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}.\, R(\overline{x}, \overline{w})) \in \Gamma \setminus E \end{array}}{r, i-1 \vdash_u \forall \overline{x}, \overline{z}.\, (S(\overline{x}, \overline{z}) \Rightarrow \overline{x} \neq \overline{v}) \vee \exists \overline{y}.\, (R(\overline{v}, \overline{y}) \wedge \overline{y} \neq \overline{w})} \text{ Learn } \texttt{DELETE} \text{ - ID - 1}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{DELETE}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad Ex(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\ r, i-1 \vdash_u \psi \qquad r, i \vdash_u \neg\psi \end{array}}{r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))]} \text{ Learn } \texttt{DELETE} \text{ Backward - 1}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{DELETE}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad Ex(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\ r, i-1 \vdash_u \psi \qquad r, 1 \vdash_u \neg\psi \end{array}}{r, i-1 \vdash_u R(\overline{t})} \text{ Learn } \texttt{DELETE} \text{ Backward - 2}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle op, u'', pr, u' \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad Ex(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\ u', u'' \in U \qquad op \in \{\oplus, \oplus^*, \ominus\} \qquad \mathit{last}(r^{i-1}).sec \neq \mathit{last}(r^i).sec \end{array}}{r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))]} \text{ Learn } \texttt{GRANT/REVOKE} \text{ Backward}$$

Figure 30: Extracting knowledge from triggers

$$\frac{\begin{array}{c} r, i-1 \vdash_u \psi \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, stmt \rangle, tr \rangle \quad Ex(s) = \emptyset \\ secEx(s) = \bot \quad t = \langle id, ow, ev, R, \phi, act, m \rangle \quad \mathit{reviseBelief}(r^{i-1}, \psi, r^i) = \top \end{array}}{r, i \vdash_u \psi} \text{ Propagate Forward Trigger Action}$$

$$\frac{\begin{array}{c} r, i \vdash_u \psi \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, stmt \rangle, tr \rangle \quad Ex(s) = \emptyset \\ secEx(s) = \bot \quad t = \langle id, ow, ev, R, \phi, act, m \rangle \quad \mathit{reviseBelief}(r^{i-1}, \psi, r^i) = \top \end{array}}{r, i-1 \vdash_u \psi} \text{ Propagate Backward Trigger Action}$$

$$\frac{\begin{array}{c} r, i-1 \vdash_u \psi \quad r, i-1 \vdash_u R(\overline{t}) \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{INSERT}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \quad Ex(s) = \emptyset \\ secEx(s) = \bot \quad t = \langle id, ow, ev, R', \phi, act, m \rangle \end{array}}{r, i \vdash_u \psi} \text{ Propagate Forward } \texttt{INSERT} \text{ Trigger Action}$$

$$\frac{\begin{array}{c} r, i-1 \vdash_u \psi \quad r, i-1 \vdash_u \neg R(\overline{t}) \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{DELETE}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \quad Ex(s) = \emptyset \\ secEx(s) = \bot \quad t = \langle id, ow, ev, R', \phi, act, m \rangle \end{array}}{r, i \vdash_u \psi} \text{ Propagate Forward } \texttt{DELETE} \text{ Trigger Action}$$

$$\frac{\begin{array}{c} r, i \vdash_u \psi \quad r, i-1 \vdash_u R(\overline{t}) \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{INSERT}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \quad Ex(s) = \emptyset \\ secEx(s) = \bot \quad t = \langle id, ow, ev, R', \phi, act, m \rangle \end{array}}{r, i-1 \vdash_u \psi} \text{ Propagate Backward } \texttt{INSERT} \text{ Trigger Action}$$

$$\frac{\begin{array}{c} r, i \vdash_u \psi \quad r, i-1 \vdash_u \neg R(\overline{t}) \quad 1 < i \leq |r| \quad r^i = r^{i-1} \cdot t \cdot s \quad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, when, \langle \langle u', \texttt{DELETE}, R, \overline{t} \rangle, \top, \top, \emptyset \rangle \rangle, tr \rangle \quad Ex(s) = \emptyset \\ secEx(s) = \bot \quad t = \langle id, ow, ev, R', \phi, act, m \rangle \end{array}}{r, i-1 \vdash_u \psi} \text{ Propagate Backward } \texttt{DELETE} \text{ Trigger Action}$$

Figure 31: Rules for propagating knowledge through triggers

$$
\frac{\begin{array}{c}
1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\
s = \langle db, U, sec, T, V, h, \langle t, \mathit{when}, \mathit{stmt} \rangle, tr \rangle \\
\mathit{secEx}(s) = \bot \qquad \mathit{Ex}(s) = \emptyset \qquad t = \langle id, ow, ev, R, \phi, act, m \rangle \\
\mathit{getAction}(act, \mathit{user}(\mathit{last}(r^{i-1}), t), \mathit{tpl}(\mathit{last}(r^{i-1}))) = \langle u', \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q}) \rangle \\
r, i-1 \vdash_u \exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w} \\
\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. \, (R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}' \in \Gamma
\end{array}}{r, i-1 \vdash_u \neg \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))]}
\quad \begin{array}{c}\text{Trigger}\\ \text{FD}\\ \texttt{INSERT}\\ \text{Disabled}\\ \text{Backward}\end{array}
$$

$$
\frac{\begin{array}{c}
1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\
s = \langle db, U, sec, T, V, h, \langle t, \mathit{when}, \mathit{stmt} \rangle, tr \rangle \\
\mathit{secEx}(s) = \bot \qquad \mathit{Ex}(s) = \emptyset \qquad t = \langle id, ow, ev, R, \phi, act, m \rangle \\
\mathit{getAction}(act, \mathit{user}(\mathit{last}(r^{i-1}), t), \mathit{tpl}(\mathit{last}(r^{i-1}))) = \langle u', \texttt{INSERT}, R, (\overline{v}, \overline{w}) \rangle \\
r, i-1 \vdash_u \forall \overline{x}, \overline{y}. \, S(\overline{x}, \overline{y}) \Rightarrow \overline{x} \neq \overline{v} \qquad \forall \overline{x}, \overline{z}. \, R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. \, S(\overline{x}, \overline{w}) \in \Gamma
\end{array}}{r, i-1 \vdash_u \neg \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))]}
\quad \begin{array}{c}\text{Trigger}\\ \text{ID}\\ \texttt{INSERT}\\ \text{Disabled}\\ \text{Backward}\end{array}
$$

$$
\frac{\begin{array}{c}
1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\
s = \langle db, U, sec, T, V, h, \langle t, \mathit{when}, \mathit{stmt} \rangle, tr \rangle \\
\mathit{secEx}(s) = \bot \qquad \mathit{Ex}(s) = \emptyset \qquad t = \langle id, ow, ev, R, \phi, act, m \rangle \\
\mathit{getAction}(act, \mathit{user}(\mathit{last}(r^{i-1}), t), \mathit{tpl}(\mathit{last}(r^{i-1}))) = \langle u', \texttt{DELETE}, R, (\overline{v}, \overline{w}) \rangle \\
r, i-1 \vdash_u \exists \overline{z}. \, S(\overline{v}, \overline{z}) \wedge \forall \overline{y}. \, (R(\overline{x}, \overline{y}) \Rightarrow \overline{y} = \overline{w}) \qquad \forall \overline{x}, \overline{z}. \, S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. \, R(\overline{x}, \overline{w}) \in \Gamma
\end{array}}{r, i-1 \vdash_u \neg \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))]}
\quad \begin{array}{c}\text{Trigger}\\ \text{ID}\\ \texttt{DELETE}\\ \text{Disabled}\\ \text{Backward}\end{array}
$$

$$
\frac{\begin{array}{c}
1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\
s = \langle db, U, sec, T, V, h, \langle t, \mathit{when}, \mathit{stmt} \rangle, tr \rangle \\
\mathit{secEx}(s) = \bot \qquad \mathit{Ex}(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\
\mathit{getAction}(act, \mathit{user}(\mathit{last}(r^{i-1}), t), \mathit{tpl}(\mathit{last}(r^{i-1}))) = \langle op, u'', p, u' \rangle \\
u', u'' \in U \quad op \in \{\oplus, \oplus^*\} \quad \langle op, u'', p, u' \rangle \notin \mathit{last}(r^{i-1}).sec \quad \mathit{last}(r^{i-1}).sec = sec
\end{array}}{r, i-1 \vdash_u \neg \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))]}
\quad \begin{array}{c}\text{Trigger}\\ \texttt{GRANT}\\ \text{Disabled}\\ \text{Backward}\end{array}
$$

$$
\frac{\begin{array}{c}
1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\
s = \langle db, U, sec, T, V, h, \langle t, \mathit{when}, \mathit{stmt} \rangle, tr \rangle \\
\mathit{secEx}(s) = \bot \qquad \mathit{Ex}(s) = \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\
\mathit{getAction}(act, \mathit{user}(\mathit{last}(r^{i-1}), t), \mathit{tpl}(\mathit{last}(r^{i-1}))) = \langle \ominus, u'', p, u' \rangle \\
u', u'' \in U \quad op \in \{\oplus, \oplus^*\} \quad \langle op, u'', p, u' \rangle \in \mathit{last}(r^{i-1}).sec \quad \mathit{last}(r^{i-1}).sec = sec
\end{array}}{r, i-1 \vdash_u \neg \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))]}
\quad \begin{array}{c}\text{Trigger}\\ \texttt{REVOKE}\\ \text{Disabled}\\ \text{Backward}\end{array}
$$

Figure 32: Extracting knowledge from triggers

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, \mathit{when}, \langle \langle u', \mathtt{INSERT}, R, \overline{t} \rangle, \top, \top, E \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\ (\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. ((R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}') \in Ex(s) \quad \overline{t} = (\overline{v}, \overline{w}, \overline{q}) \end{array}}{r, i-1 \vdash_u \exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}} \ \begin{array}{c}\text{Trigger} \\ \mathtt{INSERT} \\ \text{FD} \\ \text{Exception}\end{array}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, \mathit{when}, \langle \langle u', \mathtt{INSERT}, R, \overline{t} \rangle, \top, \top, E \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\ (\forall \overline{x}, \overline{z}. (R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w})) \in Ex(s) \qquad \overline{t} = (\overline{v}, \overline{w}) \end{array}}{r, i-1 \vdash_u \forall \overline{x}, \overline{y}. S(\overline{x}, \overline{y}) \Rightarrow \overline{x} \neq \overline{v}} \ \begin{array}{c}\text{Trigger} \\ \mathtt{INSERT} \\ \text{ID} \\ \text{Exception}\end{array}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, \mathit{when}, \langle \langle u', \mathtt{DELETE}, R, \overline{t} \rangle, \top, \top, E \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \\ (\forall \overline{x}, \overline{z}. (S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. R(\overline{x}, \overline{w})) \in Ex(s) \qquad \overline{t} = (\overline{v}, \overline{w}) \end{array}}{r, i-1 \vdash_u \exists \overline{z}. S(\overline{v}, \overline{z}) \wedge \forall \overline{y}. (R(\overline{v}, \overline{y}) \Rightarrow \overline{y} = \overline{w})} \ \begin{array}{c}\text{Trigger} \\ \mathtt{DELETE} \\ \text{ID} \\ \text{Exception}\end{array}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, \langle \langle u', \mathtt{SELECT}, \phi[\overline{x} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))] \rangle, \top, \top, \emptyset \rangle, \mathit{stmt}, tr \rangle \\ \mathit{secEx}(s) = \top \vee Ex(s) \neq \emptyset \qquad t = \langle id, ow, ev, R, \phi, act, m \rangle \end{array}}{r, i-1 \vdash_u \phi[\overline{x}^{|R'|} \mapsto \mathit{tpl}(\mathit{last}(r^{i-1}))]} \ \begin{array}{c}\text{Trigger} \\ \text{Exception}\end{array}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, \langle \langle u', \mathtt{SELECT}, \phi \rangle, \top, \top, \emptyset \rangle, \langle \langle u', \mathtt{INSERT}, R, \overline{t} \rangle, res, aC, E \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad Ex(s) \neq \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \end{array}}{r, i-1 \vdash_u \neg R(\overline{t})} \ \begin{array}{c}\text{Trigger} \\ \mathtt{INSERT} \\ \text{Exception}\end{array}$$

$$\frac{\begin{array}{c} 1 < i \leq |r| \qquad r^i = r^{i-1} \cdot t \cdot s \qquad \mathit{invoker}(\mathit{last}(r^{i-1})) = u \\ s = \langle db, U, sec, T, V, h, \langle t, \langle \langle u', \mathtt{SELECT}, \phi \rangle, \top, \top, \emptyset \rangle, \langle \langle u', \mathtt{DELETE}, R, \overline{t} \rangle, res, aC, E \rangle, tr \rangle \\ \mathit{secEx}(s) = \bot \qquad Ex(s) \neq \emptyset \qquad t = \langle id, ow, ev, R', \phi, act, m \rangle \end{array}}{r, i-1 \vdash_u R(\overline{t})} \ \begin{array}{c}\text{Trigger} \\ \mathtt{DELETE} \\ \text{Exception}\end{array}$$

$$\frac{\begin{array}{c} n+1 < i \leq |r| \qquad s_1, s_2, \ldots, s_n \in \Omega_M \qquad t_1, \ldots, t_n \in \mathcal{TRIGGER}_D \\ \mathit{secEx}(s_n) = \top \vee Ex(s_n) \neq \emptyset \quad r^i = r^{i-n-1} \cdot \langle u, \mathtt{INSERT}, R, \overline{t} \rangle \cdot s_1 \cdot t_1 \cdot s_2 \cdot \ldots \cdot t_n \cdot s_n \\ s_n = \langle db, U, sec, T, V, h, \langle t_n, \mathit{when}, \mathit{stmt} \rangle, \langle \epsilon, \epsilon, \epsilon, \epsilon \rangle \rangle \end{array}}{r, i \vdash_u \neg R(\overline{t})} \ \begin{array}{c}\text{Trigger} \\ \text{Rollback} \\ \mathtt{INSERT}\end{array}$$

$$\frac{\begin{array}{c} n+1 < i \leq |r| \qquad s_1, s_2, \ldots, s_n \in \Omega_M \qquad t_1, \ldots, t_n \in \mathcal{TRIGGER}_D \\ \mathit{secEx}(s_n) = \top \vee Ex(s_n) \neq \emptyset \quad r^i = r^{i-n-1} \cdot \langle u, \mathtt{DELETE}, R, \overline{t} \rangle \cdot s_1 \cdot t_1 \cdot s_2 \cdot \ldots \cdot t_n \cdot s_n \\ s_n = \langle db, U, sec, T, V, h, \langle t_n, \mathit{when}, \mathit{stmt} \rangle, \langle \epsilon, \epsilon, \epsilon, \epsilon \rangle \rangle \end{array}}{r, i \vdash_u R(\overline{t})} \ \begin{array}{c}\text{Trigger} \\ \text{Rollback} \\ \mathtt{DELETE}\end{array}$$

Figure 33: Extracting knowledge from trigger's exceptions

C. DATABASE INTEGRITY

In this section, we present the formal definition of the $\rightsquigarrow_{auth}$ relation, which is used to define database integrity. Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP. We denote by $\mathcal{VIEW}_D^{owner}$ the set of all D-views with the owner's privileges, i.e., $\mathcal{VIEW}_D^{owner} = \{\langle V, o, q, m\rangle \in \mathcal{VIEW}_D \mid m = O\}$, and by $\mathcal{PRIV}_D^{\texttt{SELECT},\mathcal{VIEW}_D^{owner}}$ the set of privileges $\{pr \in \mathcal{PRIV}_D \mid pr = \langle \texttt{SELECT}, V\rangle \wedge V \in \mathcal{VIEW}_D^{owner}\}$. Given a state an M-state $s = \langle db, U, sec, T, V, c\rangle$ and a revoke command $r = \langle \ominus, u, p, u'\rangle$, we denote by $applyRev(s, r)$ the state $\langle db, U, revoke(sec, u, p, u'), T, V, c\rangle$ obtained by executing the `REVOKE` command. Given a system's configuration $M = \langle D, \Gamma\rangle$, a query q, a set of views V with owner's privileges, and a set of tables T, we say that V and T determine q, denoted by $determines_M(T, V, q)$, iff for all $db \in \Omega_D^\Gamma$, for all db_1, $db_2 \in [\![db]\!]_{V,T}$, $[q]^{db_1} = [q]^{db_2}$, where $[\![db]\!]_{V,T}$ denotes the set $\{db' \in \Omega_D^\Gamma \mid \forall T_1 \in T. T_1(db) = T_1(db') \wedge \forall V_1 \in V. V_1(db) = V_1(db')\}$. Further details on the concept of determinacy can be found in [34]. Finally, the relation $\rightsquigarrow_{auth} \subseteq \Omega_M \times (\mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D)$ is the smallest relation satisfying the inference rules given in Figure 34.

$$\frac{u, u' \in U \quad R \in D \quad \overline{t} \in \mathbf{dom}^{|R|} \quad g = \langle op, u, \langle op', R\rangle, u'\rangle \quad g \in sec \quad \langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} g \quad op' \in \{\mathtt{INSERT}, \mathtt{DELETE}\}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle u, op', R, \overline{t}\rangle}\ \begin{array}{c}\mathtt{INSERT}\\ \mathtt{DELETE}\end{array}$$

$$\frac{u, u' \in U \quad v \in \mathcal{VIEW}_D \quad g = \langle op, u, \langle \mathtt{CREATE\ VIEW}\rangle, u'\rangle \quad g \in sec \quad \langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} g}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle u, \mathtt{CREATE}, v\rangle}\ \begin{array}{c}\mathtt{CREATE}\\ \mathtt{VIEW}\end{array}$$

$$\frac{u, u' \in U \quad t = \langle id, ow, ev, R, \phi, stmt, m\rangle \quad t \in \mathcal{TRIGGER}_D \quad g = \langle op, u, \langle \mathtt{CREATE\ TRIGGER}, R\rangle, u'\rangle \quad g \in sec \quad \langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} g}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle u, \mathtt{CREATE}, t\rangle}\ \begin{array}{c}\mathtt{CREATE}\\ \mathtt{TRIGGER}\end{array}$$

$$\frac{R \in D \quad \overline{t} \in \mathbf{dom}^{|R|} \quad op' \in \{\mathtt{INSERT}, \mathtt{DELETE}\}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle admin, op', R, \overline{t}\rangle}\ \begin{array}{c}\mathtt{INSERT}\\ \mathtt{DELETE}\\ \text{admin}\end{array}$$

$$\frac{v \in \mathcal{VIEW}_D \quad v = \langle id, admin, q, m\rangle}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle admin, \mathtt{CREATE}, v\rangle}\ \begin{array}{c}\mathtt{CREATE}\\ \mathtt{VIEW}\\ \text{admin}\end{array}$$

$$\frac{t = \langle id, admin, ev, R, \phi, stmt, m\rangle \quad t \in \mathcal{TRIGGER}_D}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle admin, \mathtt{CREATE}, t\rangle}\ \begin{array}{c}\mathtt{CREATE}\\ \mathtt{TRIGGER}\\ \text{admin}\end{array}$$

$$\frac{u, u' \in U \quad priv \in \mathcal{PRIV}_D \quad s = \langle db, U, sec, T, V, c\rangle \quad s' = \langle db, U, sec', T, V, c\rangle \quad s' = applyRev(s, \langle \ominus, u, p, u'\rangle) \quad \forall g \in sec'.\, s' \rightsquigarrow_{auth} g}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle \ominus, u, priv, u'\rangle}\ \mathtt{REVOKE}$$

$$\frac{u, u', u'' \in U \quad op \in \{\oplus, \oplus^*\} \quad priv \in \mathcal{PRIV}_D \quad g = \langle \oplus^*, u', priv, u''\rangle \quad g \in sec \quad \langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} g}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle op, u, priv, u'\rangle}\ \mathtt{GRANT\text{-}1}$$

$$\frac{u \in U \quad op \in \{\oplus, \oplus^*\} \quad priv \in \mathcal{PRIV}_D \setminus \mathcal{PRIV}_D^{\mathtt{SELECT}, \mathcal{VIEW}^{owner}{}_D}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle op, u, priv, admin\rangle}\ \mathtt{GRANT\text{-}2}$$

$$\frac{\begin{array}{c} u, owner \in U \quad op \in \{\oplus, \oplus^*\} \quad priv = \langle \mathtt{SELECT}, v\rangle \\ v = \langle id, owner, q, O\rangle \quad v \in V \quad V' \subseteq V \cap \mathcal{VIEW}_D^{owner} \\ T' \subseteq D \quad determines_M(T', V', q) \\ hasAccess(\langle db, U, sec, T, V, c\rangle, V' \cup T', owner, \oplus^*)\end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle op, u, priv, owner\rangle}\ \mathtt{GRANT\text{-}3}$$

$$\frac{\begin{array}{c} u, owner \in U \quad op \in \{\oplus, \oplus^*\} \quad priv = \langle \mathtt{SELECT}, v\rangle \\ v = \langle id, owner, q, O\rangle \quad v \in V \quad owner \neq admin \\ V' \subseteq V \cap \mathcal{VIEW}_D^{owner} \quad T' \subseteq D \quad determines_M(T', V', q) \\ hasAccess(\langle db, U, sec, T, V, c\rangle, V' \cup T', owner, \oplus)\end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle op, u, priv, admin\rangle}\ \mathtt{GRANT\text{-}4}$$

$$\frac{u, owner \in U \quad op \in \{\oplus, \oplus^*\} \quad v \in V \quad priv = \langle \mathtt{SELECT}, v\rangle \quad v = \langle id, owner, q, A\rangle}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle op, u, priv, owner\rangle}\ \mathtt{GRANT\text{-}5}$$

$$\frac{u \in \mathcal{U} \quad u' = admin}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle u', \mathtt{ADD\_USER}, u\rangle}\ \mathtt{ADD\ USER}$$

$$\frac{\begin{array}{c} t = \langle id, ow, ev, R, \phi, stmt, O\rangle \quad t \in T \\ \langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} getAction(stmt, ow, tpl(c)) \\ [\phi[\overline{x}^{|R|} \mapsto tpl(c)]]^{db} = \top \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} t}\ \begin{array}{c}\mathtt{EXECUTE}\\ \mathtt{TRIGGER\text{-}1}\end{array}$$

$$\frac{\begin{array}{c} t = \langle id, ow, ev, R, \phi, stmt, A\rangle \quad t \in T \\ \langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} getAction(stmt, invoker(c), tpl(c)) \\ \langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} getAction(stmt, ow, tpl(c)) \\ [\phi[\overline{x}^{|R|} \mapsto tpl(c)]]^{db} = \top \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} t}\ \begin{array}{c}\mathtt{EXECUTE}\\ \mathtt{TRIGGER\text{-}2}\end{array}$$

$$\frac{u \in U \quad q \in RC}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle u, \mathtt{SELECT}, q\rangle}\ \mathtt{SELECT}$$

$$\frac{t = \langle id, ow, ev, R, \phi, stmt, m\rangle \quad t \in T \quad [\phi[\overline{x}^{|R|} \mapsto tpl(c)]]^{db} = \bot}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} t}\ \begin{array}{c}\mathtt{EXECUTE}\\ \mathtt{TRIGGER\text{-}3}\end{array}$$

$$hasAccess(\langle db, U, sec, T, V, c\rangle, S, u, op) = \begin{cases} \top & \text{if } u \neq admin \wedge \forall v \in S.\, \exists u'' \in U, g \in sec, op' \in \{op, \oplus^*\}. \\ & \quad g = \langle op', u, \langle \mathtt{SELECT}, v\rangle, u''\rangle \wedge \langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} g \\ \top & \text{if } u = admin \wedge \forall v \in S.\, \exists u'' \in U, op' \in \{op, \oplus^*\}. \\ & \quad \langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} \langle op', u, \langle \mathtt{SELECT}, v\rangle, u''\rangle \\ \bot & otherwise \end{cases}$$

Figure 34: Definition of the $\rightsquigarrow_{auth}$ relation

D. DATA CONFIDENTIALITY

In this section, we define indistinguishability of runs. We first formalize the notion of u-projection. Afterwards, we define the notion of consistency between u-projections. Finally, we formalize the indistinguishability relation $\cong_{P,u}$.

We recall that, given a run r, we denote by r^i, where $1 \leq i \leq |r|$, the prefix of r obtained by truncating r at the i-th state. In the rest of the paper, we use r^0 to denote the empty run.

D.1 Projections

Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ and f is an M-PDP, L be the P-LTS, and u be a user in $\mathcal{U}$. Given a run $r \in \mathit{traces}(L)$, its *u-projection*, denoted by $r|_u$, is obtained by (1) replacing each action not issued by u with $*$, (2) replacing each trigger whose invoker is not u with $*$, and (3) replacing all non-empty sequences of $*$-transitions with a single $*$-transition. Note that the $*$-transitions in the u-projections represent whether u's actions are executed consecutively or not. With a slight abuse of notation, we extend all the notation we use for runs also to u-projections. For instance, $r|_u^i$ denotes the prefix obtained by truncating $r|_u$ at its i-th state. Formally, the u-projection $r|_u$ is defined as $c(v(r, u))$. The function v takes as input a run r and a user u and returns another run in which all non-u actions are replaced with $*$.

$$v(r,u) = \begin{cases} v(r^{|r|-1}, u) \cdot a \cdot s & \text{if } r = r^{|r|-1} \cdot a \cdot s \text{ and } s \in \Omega_M \\ & \text{and } a \in \mathcal{A}_{D,u} \text{ and } |r| > 1 \\ v(r^{|r|-1}, u) \cdot * \cdot s & \text{if } r = r^{|r|-1} \cdot a \cdot s \text{ and } s \in \Omega_M \\ & \text{and } a \in \mathcal{A}_{D,u'} \text{ and } u' \neq u \text{ and} \\ & |r| > 1 \\ v(r^{|r|-1}, u) \cdot t \cdot s & \text{if } r = r^{|r|-1} \cdot t \cdot s \text{ and } s \in \Omega_M \\ & \text{and } t \in \mathcal{TRIGGER}_D \text{ and} \\ & \mathit{invoker}(\mathit{last}(r^{|r|-1})) = u \text{ and} \\ & |r| > 1 \\ v(r^{|r|-1}, u) \cdot * \cdot s & \text{if } r = r^{|r|-1} \cdot t \cdot s \text{ and } s \in \Omega_M \\ & \text{and } t \in \mathcal{TRIGGER}_D \text{ and} \\ & \mathit{invoker}(\mathit{last}(r^{|r|-1})) \neq u \text{ and} \\ & |r| > 1 \\ s & \text{if } r = s \text{ and } s \in \Omega_M \end{cases}$$

The function c takes as input a run r containing $*$-transitions and replaces each sequence of $*$-transitions with a single $*$-transition. Note that the function c is obtained by repeatedly applying the function c' until the computation reaches a fixed point. The function c' is as follows:

$$c'(r) = \begin{cases} c'(r^{|r|-1}) \cdot a \cdot s & \text{if } r = r^{|r|-1} \cdot a \cdot s \text{ and } a \neq * \\ & \text{and } s \in \Omega_M \text{ and } |r| > 1 \\ c'(r^{|r|-2}) \cdot * \cdot s & \text{if } r = r^{|r|-2} \cdot * \cdot s' \cdot * \cdot s \text{ and} \\ & s, s' \in \Omega_M \text{ and } |r| > 2 \\ s & \text{if } r = s \text{ and } s \in \Omega_M \\ r & \text{if } r = s \cdot * \cdot s' \text{ and } s, s' \in \Omega_M \end{cases}$$

D.2 Consistency

Before defining the notion of consistency, we define the function *labels* which takes as input a run r and returns as output the sequence of labels in the run. In more detail, $\mathit{labels}(r)$ is obtained from r by dropping all the states. We now define the notion of consistency between two u-projections.

Definition D.1. Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ and f is an M-PDP, L be the P-LTS, and u be a user in $\mathcal{U}$. Furthermore, let $r|_u$ and $r'|_u$ be two u-projections for the runs r and r' in $\mathit{traces}(L)$. We say that $r|_u$ *and* $r'|_u$ *are consistent* iff the following conditions hold:

1. $|r|_u| = |r'|_u|$.
2. $\mathit{labels}(r|_u) = \mathit{labels}(r'|_u)$.
3. $\mathit{triggers}(\mathit{last}(r|_u)) = \epsilon$ iff $\mathit{triggers}(\mathit{last}(r'|_u)) = \epsilon$.
4. for all i such that $1 \leq i \leq |r|_u|$, if $r|_u^i = r|_u^{i-1} \cdot a \cdot s$ and $a \neq *$, then
   - $\mathit{res}(\mathit{last}(r|_u^i)) = \mathit{res}(\mathit{last}(r'|_u^i))$,
   - $\mathit{secEx}(\mathit{last}(r|_u^i)) = \mathit{secEx}(\mathit{last}(r'|_u^i))$,
   - if a is a trigger, then $\mathit{acC}(\mathit{last}(r|_u^i)) = \mathit{acC}(\mathit{last}(r'|_u^i))$,
   - $\mathit{invoker}(\mathit{last}(r|_u^i)) = \mathit{invoker}(\mathit{last}(r'|_u^i))$,
   - $\mathit{triggers}(\mathit{last}(r|_u^i)) = \mathit{triggers}(\mathit{last}(r'|_u^i))$,
   - $\mathit{tpl}(\mathit{last}(r|_u^i)) = \mathit{tpl}(\mathit{last}(r'|_u^i))$,
   - and $\mathit{Ex}(\mathit{last}(r|_u^i)) = \mathit{Ex}(\mathit{last}(r'|_u^i))$. $\square$

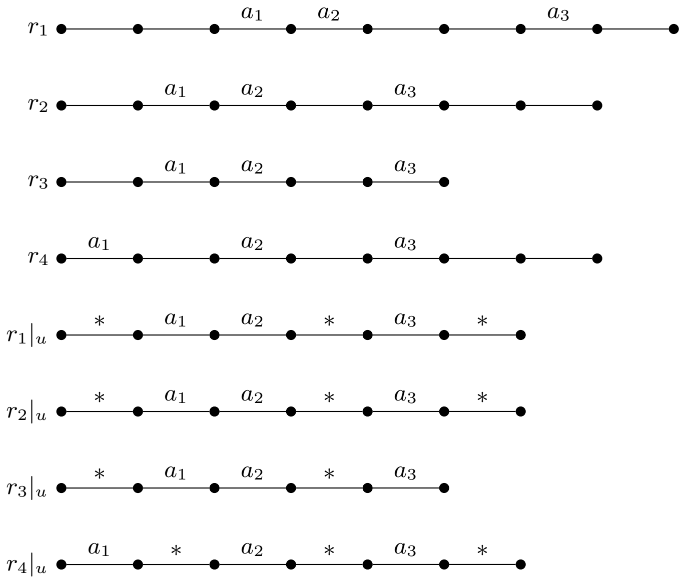


Figure 35: The runs r_1, r_2, r_3, and r_4, where the states are represented using black dots, the actions a_1, a_2, and a_3 issued by the user u are written above the edges connecting the states, and the actions of the other users are omitted. The u-projections of these runs are, respectively, $r_1|_u$, $r_2|_u$, $r_3|_u$, and $r_4|_u$. The runs r_1 and r_2 have u-projections with the same labels, whereas the runs r_3 and r_4 have u-projections with different labels.

Figure 35 depicts four runs. The states are represented just as black dots and the action between two states is written above the edge connecting them. Note that we represent just the actions a_1, a_2, and a_3 issued by the user u. Assume that (a) the action's effects are the same in all the runs and (b) the *invoker*, *res*, *secEx*, *triggers*, *tpl*, and *Ex* functions return the same results in all runs. It is easy to see that $r_1|_u$ and $r_2|_u$ are consistent projections, whereas $r_3|_u$ and $r_4|_u$ are not. Furthermore, there is no other pair of consistent u-projections between the runs in the figure.

D.3 Indistinguishability

Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be an M-partial state, and $u \in U$ be a user. The set

$permissions(s,u)$ is $permissions(s,u) := \{\langle \oplus, \texttt{SELECT}, O\rangle \mid \exists u' \in U, op \in \{\oplus, \oplus^*\}. \langle op, u, \langle \texttt{SELECT}, O\rangle, u'\rangle \in sec\}$. Note that $permissions(s, admin) = D \cup V$ since the administrator has read access to the whole database. We extend permissions to M-states as follows. Given an M-state $s' = \langle db, U, sec, T, V, c\rangle$, $permissions(s', u) = permissions(\langle db, U, sec, T, V\rangle, u)$.

We are now ready to introduce the notion of indistinguishability between two runs. Intuitively, two runs r and r' are indistinguishable for a user u iff (1) their u-projections are consistent, and (2) for each action of the user u as well as for the last states in the two runs, the policy, the triggers, the views, the users, and the data disclosed by the policy are the same in r and r'.

Definition D.2. Let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, and u be a user.

We say that two runs r and r' in $traces(L)$ are (P,u)-*indistinguishable*, written $r \cong_{u,P} r'$, iff

1. $r|_u$ and $r'|_u$ are consistent,
2. $pState(last(r))$ and $pState(last(r'))$ are (M,u)-data indistinguishable, and
3. for all i such that $1 \leq i \leq |r|_u| - 1$, if $r|_u^{i+1} = r|_u^i \cdot a \cdot s$, $a \neq *$, and $s \in \Omega_M$, then $pState(last(r|_u^i))$ and $pState(last(r'|_u^i))$ are (M,u)-data indistinguishable.

We say that two M-partial states $s = \langle db, U, sec, T, V\rangle$ and $s' = \langle db', U', sec', T', V'\rangle$ are (M,u)-*data indistinguishable*, written $s \cong_{u,M}^{data} s'$, iff

1. $U = U'$,
2. $sec = sec'$,
3. $T = T'$,
4. $V = V'$,
5. for all relation schema $R \in D$ for which $\langle \oplus, \texttt{SELECT}, R\rangle \in permissions(s,u)$, $db(R) = db'(R)$, and
6. for all views $v \in \mathcal{VIEW}_D^{owner}$ for which $\langle \oplus, \texttt{SELECT}, v\rangle \in permissions(s,u)$, $db(v) = db'(v)$. $\square$

PROPOSITION D.1. *Let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, and $u \in \mathcal{U}$ be a user. The indistinguishability relation $\cong_{P,u}$ is an equivalence relation over $traces(L)$.*

PROOF. We now prove that $\cong_{P,u}$ is reflexive, symmetric, and transitive. This implies the fact that $\cong_{P,u}$ is an equivalence relation over $traces(L)$. In the following, let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, and $u \in \mathcal{U}$ be a user. From the definition of data indistinguishability and the results in [24], it follows that the data-indistinguishability relation $\cong_{u,M}^{data}$ is an equivalence relation over the set of all partial states.

Reflexivity Let $r \in traces(L)$ be a run. It follows trivially that $r|_u = r|_u$. From this, it follows that $r|_u$ and $r|_u$ are consistent. It is easy to see that r is indistinguishable from r. Indeed, the database states are the same in r and r and the data-indistinguishability relation is reflexive [24].

Symmetry Let $r, r' \in traces(L)$ be two runs such that $r \cong_{P,u} r'$. From this, it follows that $r|_u$ and $r'|_u$ are consistent. Note that the consistency definition is symmetric. Therefore, also $r'|_u$ and $r|_u$ are consistent. From this and the symmetry of data indistinguishability [24], it follows the symmetry of $\cong_{P,u}$.

Transitivity Let $r, r', r'' \in traces(L)$ be three runs such that $r \cong_{P,u} r'$ and $r' \cong_{P,u} r''$. From this it follows that $r|_u$ and $r'|_u$ are consistent and $r'|_u$ and $r''|_u$ are consistent. It is easy to see that also $r|_u$ and $r''|_u$ are consistent. From this and the transitivity of data indistinguishability [24], it follows the transitivity of $\cong_{P,u}$. $\square$

Given a run r, we denote by $[\![r]\!]_{P,u}$ the equivalence class of r defined by $\cong_{P,u}$ over $traces(L)$. Similarly, we denote by $[\![s]\!]_{u,M}^{data}$ the equivalence class of s defined by $\cong_{u,M}^{data}$ over Π_M.

E. ENFORCING DATABASE INTEGRITY

In this section, we first define the access control function f_{int}, which models the f_{int} procedure described in §6. Afterwards, we prove that the function f_{int} satisfies the database integrity property. Finally, we prove that the data complexity of f_{int} is AC^0.

The function f_{int} is as follows:

$$f_{int}(s,a) = \begin{cases} \top & \text{if } \mathit{trigger}(s) = \epsilon \wedge s \rightsquigarrow^{appr}_{auth} a \\ \top & \text{if } \mathit{trigger}(s) = t \wedge t \neq \epsilon \wedge a = \mathit{trigCond}(s) \\ \top & \text{if } \mathit{trigger}(s) = t \wedge t \neq \epsilon \wedge a = \mathit{trigAct}(s) \wedge \\ & \quad s \rightsquigarrow^{appr}_{auth} t \\ \bot & \text{otherwise} \end{cases}$$

The function $\mathit{trigCond}(s)$ (respectively $\mathit{trigAct}(s)$) returns the condition (respectively the action) associated with the trigger $\mathit{trigger}(s)$. If $\mathit{trigger}(s) = \langle id, ow, e, R, \phi, st, O\rangle$, then $\mathit{trigAct}(s) = \mathit{getAction}(st, ow, \mathit{tpl}(s))$ and $\mathit{trigCond}(s) = \langle ow, \texttt{SELECT}, \phi[\overline{x}^{|R|} \mapsto \mathit{tpl}(s)]\rangle$. If $\mathit{trigger}(s) = \langle id, ow, e, R, \phi, st, A\rangle$, then $\mathit{trigAct}(s) = \mathit{getAction}(st, \mathit{invoker}(s), \mathit{tpl}(s))$ and $\mathit{trigCond}(s) = \langle \mathit{invoker}(s), \texttt{SELECT}, \phi[\overline{x}^{|R|} \mapsto \mathit{tpl}(s)]\rangle$.

Recall that, given an M-state $s = \langle db, U, sec, T, V, c\rangle$ and a revoke statement $r = \langle \ominus, u, p, u'\rangle$, $\mathit{applyRev}(s, r)$ denotes the state $\langle db, U, \mathit{revoke}(sec, u, p, u'), T, V, c\rangle$.

The relation $\rightsquigarrow^{appr}_{auth} \subseteq \Omega_M \times (\mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D)$ is the smallest relation satisfying the inference rules given in Figure 37. We remark that $\rightsquigarrow^{appr}_{auth}$ is a sound and computable under-approximation of the relation $\rightsquigarrow_{auth}$. In the rules, we use a number of auxiliary functions. The most important ones are:

(a) the aT (respectively aV) function that takes as input a database state, an operator op in $\{\oplus, \oplus^*\}$, and a user, and returns the set of tables (respectively views) that the user is authorized to read (if $op = \oplus$) or to delegate the read access (if $op = \oplus^*$) according to our approximation of $\rightsquigarrow_{auth}$, and

(b) the $\mathit{apprDet}$ function is used to determine whether a set of tables and a set of views completely determine the result of a formula ϕ in all possible database states. Note that the function $\mathit{apprDet}$ is a sound under-approximation of the concept of *query determinacy* [34].

In the following, we define the functions *extend* and *apprDet*. The functions aT and aV are defined in Figure 37. We assume that both the formula ϕ and the set of views V in the state s contain just views with owner's privileges. This is without loss of generality. Indeed, views with activator's privileges are just syntactic sugar, they do not disclose additional information to a user u other than what he is already authorized to read because they are executed under u's privileges. If ϕ and s contain views with activator's privileges, we can compute another formula ϕ' and a state s' without views with activator's privileges as follows. We replace, in the formula ϕ, the predicates of the form $V(\overline{x})$, where V is a view with activator's privileges, with V's definition, and we repeat this process until the resulting formula ϕ' no longer contains views with activator's privileges. Similarly, the set V' is obtained from V by (1) removing all views with activator's privileges, and (2) for each view $v \in V$ with owner's privileges, replacing the predicates of the form $V(\overline{x})$ in v's definition, where V is a view with activator's privileges, with V's definition until v's definition no longer contains views with activator's privileges. The security policy sec' is also obtained from sec by removing all references to views with activator's privileges. Therefore, in §E.1-E.2 we ignore views with activator's privileges as the extension to the general case is trivial.

E.1 Extend function

We now define the *extend* function, which takes as input a system configuration M, an M-state s, and a set of views with owner's privileges, and returns a set of views V' such that $V \subseteq V'$. Given a system configuration M, an M-partial state $s = \langle db, U, sec, T, V\rangle$, and a normalized view $\langle v, o, q, O\rangle \in V$, we denote by $\mathit{inline}_M(\langle v, o, q, O\rangle, s)$ the view $\langle v, o, q', O\rangle$ where q' is obtained from q by replacing all occurrences of views in V with owner's privileges with their definitions. Note that inline_M does not compute a fixpoint, i.e., if a view's definition refers to another view, the latter is not replaced with its definition. The function $\mathit{extend}(M, s, V)$ returns the set $V \cup \{\mathit{inline}(v, s) | v \in \mathit{extend}(M, s, V)\}$.

Lemma E.1. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be an M-partial state, $V' \subseteq V$ be a set of views with owner's privileges. For each view $v \in \mathit{extend}(M, s, V')$, there is a view $v' \in V'$ such that v and v' disclose the same data.*

Proof. *Sketch:* Assume, for contradiction's sake, that there is a view $v \in \mathit{extend}(M, s, V')$ such that all the views in V' disclose different data from v. This is impossible because v has been obtained by a view $v' \in V'$ just by replacing the views with their definitions and the definitions of v and v' are semantically equivalent. $\square$

E.2 A sound under-approximation of query determinacy

The definition of the function $\mathit{apprDet}(T, V, q)$ is shown in Figure 36. Before proving that *apprDet* is a sound approximation of *determines*, we extend *determines* from sentences to formulae.

We first introduce assignments. Let **dom** be the universe and **var** be an infinite countably set of variable identifiers. An *assignment* ν is a partial function from **var** to **dom** that maps variables to values in the universe. Given a formula ϕ and an assignment ν, we say that ν *is well-formed for* ϕ iff ν is defined for all variables in $\mathit{free}(\phi)$. Given an assignment ν and a sequence of variables $\overline{x}$ such that ν is defined for each $x \in \overline{x}$, we denote by $\nu(\overline{x})$ the tuple obtained by replacing each occurrence of $x \in \overline{x}$ with $\nu(x)$. Given an assignment ν, a variable $v \in$ **var**, and a value $u \in$ **dom**, we denote by $\nu \oplus [v \mapsto u]$ the assignment ν' obtained as follows: $\nu'(v') = \nu(v')$ for any $v' \neq v$, and $\nu'(v) = u$. Finally, given a formula ϕ with free variables $\mathit{free}(\phi)$ and an assignment ν, we denote by $\phi \circ \nu$ the formula ϕ' obtained by replacing, for each free variable $x \in \mathit{free}(\phi)$ such that $\nu(x)$ is defined, all the free occurrences of x with $\nu(x)$.

Given a system's configuration $M = \langle D, \Gamma\rangle$, a formula ϕ, a set of views V with owner's privileges, a set of tables T, and a well-formed assignment ν for ϕ, we say that V and T determine (ϕ, ν), denoted by $\mathit{determines}_M(T, V, \phi, \nu)$, iff for all $db \in \Omega_D^\Gamma$, for all db_1, $db_2 \in [\![db]\!]_{V,T}$, $[\phi \circ \nu]^{db_1} = [\phi \circ \nu]^{db_2}$. In the following, given a view $\langle u, o, q, m\rangle$, we denote by $\mathit{def}(\langle u, o, q, m\rangle)$ its definition q.

In Lemma $E.2$, we show that *apprDet* is, indeed, a sound under-approximation of query determinacy.

$$
apprDet(T, V, \phi, s, M) = \begin{cases} \top & \text{if } \exists \langle v, o, q, O\rangle \in extend(M, s, V).\, q = \{\overline{x} | \phi(\overline{x})\} \\ \top & \text{if } \phi = (x = v) \vee \phi = \top \vee \phi = \bot \\ \top & \text{if } \phi = R(\overline{x}) \wedge R \in T \\ \top & \text{if } \phi = V(\overline{x}) \wedge \exists u \in \mathcal{U}, q \in RC.\, \langle V, u, q, O\rangle \in V \\ \top & \text{if } \phi = (\psi \wedge \gamma) \wedge apprDet(T, V, \psi, s, M) = \top \wedge apprDet(T, V, \gamma, s, M) = \top \\ \top & \text{if } \phi = (\psi \vee \gamma) \wedge apprDet(T, V, \psi, s, M) = \top \wedge apprDet(T, V, \gamma, s, M) = \top \\ \top & \text{if } \phi = (\neg\psi) \wedge apprDet(T, V, \psi, s, M) = \top \\ \top & \text{if } \phi = (\exists x.\, \psi) \wedge apprDet(T, V, \psi, s, M) = \top \\ \top & \text{if } \phi = (\forall x.\, \psi) \wedge apprDet(T, V, \psi, s, M) = \top \\ \bot & \text{otherwise} \end{cases}
$$

Figure 36: *apprDet* function

LEMMA E.2. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be an M-partial state, $T' \subseteq D$ be a set of tables, $V' \subseteq V$ be a set of views with owner's privileges, and ϕ be a formula. If $apprDet(T', V', \phi, s, M) = \top$, then for all well-formed assignments ν for ϕ, $determines_M(T', V', \phi, \nu)$ holds.*

PROOF. Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be an M-partial state, $T' \subseteq D$ be a set of tables, $V' \subseteq V$ be a set of views with owner's privileges, and ϕ be a formula. We prove the lemma by structural induction over the formula ϕ.

Base Case: There are a number of alternatives.

$\phi := R(\overline{x})$ Assume that $apprDet(T, V, R(\overline{x}), s, M) = \top$. There are two cases:

1. $R \in T'$. In this case, the set T' trivially determines the formula $R(\overline{x})$ for any well-formed assignment ν. Therefore, $determines_M(T', V', R(\overline{x}), \nu)$ holds. Indeed, assume that this is not the case. Thus, there are three database states db, db_1, and db_2 such that $db_1, db_2 \in [\![db]\!]_{V',T'}$ and $[R(\overline{x}) \circ \nu]^{db_1} \neq [R(\overline{x}) \circ \nu]^{db_2}$. From this and the RC semantics, it follows that $db_1(R) \neq db_2(R)$. From this, $R \in T'$, and $db_1, db_2 \in [\![db]\!]_{V',T'}$, it follows that $db_1(R) = db_2(R)$ leading to a contradiction.
2. there is a view v' in $extend(M, s, V')$ such that $def(v') = \{\overline{x} | R(\overline{x})\}$. This means that there is a sequences of views $V_1, \ldots, V_n$ in s such that $def(V_1) = \{\overline{x} | R(\overline{x})\}$, $def(V_2) = \{\overline{x} | V_1(\overline{x})\}$, $\ldots$, $def(V_n) = \{\overline{x} | V_{n-1}(\overline{x})\}$, and $V_n \in V'$. Therefore, the set V' trivially determines the formula $R(\overline{x})$ for any well-formed assignment ν, and V_n and R are equivalent. Therefore, $determines_M(T', V', R(\overline{x}), \nu)$ holds.

$\phi := V(\overline{x})$ Assume that $apprDet(T, V, V(\overline{x}), s, M) = \top$. There are two cases:

1. There is a view $\langle V, o, q, O\rangle \in V'$. In this case, the set V' trivially determines the formula $V(\overline{x})$ for any assignment ν that is well-formed for ϕ. Therefore, $determines_M(T', V', V(\overline{x}), \nu)$ holds.
2. there is a view v' in $extend(M, s, V')$ such that $def(v') = \{\overline{x} | V(\overline{x})\}$. This means that there is a sequences of views V_1, $\ldots$, V_n in s such that $def(V_1) = \{\overline{x} | V(\overline{x})\}$, $def(V_2) = \{\overline{x} | V_1(\overline{x})\}$, $\ldots$, $def(V_n) = \{\overline{x} | V_{n-1}(\overline{x})\}$, and $V_n \in V'$. Therefore, the set V' trivially determines the formula $V(\overline{x})$ for any well-formed assignment ν, and V_n and V are equivalent. Therefore, $determines_M(T', V', V(\overline{x}), \nu)$ holds.

$\phi := x = v$ For any well-formed assignment ν, the empty set trivially determines the formula $x = v$ and $apprDet(T', V', x = v, s, M) = \top$.

$\phi := \top$ The proof of this case is similar to that of $\phi := x = v$.

$\phi := \bot$ The proof of this case is similar to that of $\phi := x = v$.

This concludes the proof of the base case.

Induction Step: Assume that the claim holds for all sub-formulae of ϕ. There are a number of cases:

$\phi := \psi \wedge \gamma$ Assume that $apprDet(T', V', \psi \wedge \gamma, s, M) = \top$. There are two cases:

1. $apprDet(T', V', \psi, s, M) = \top$ and $apprDet(T', V', \gamma, s, M) = \top$. From the induction hypothesis, it follows that both $determines_M(T', V', \psi, \nu)$ and $determines_M(T', V', \gamma, \nu)$ hold for all well-formed assignments ν. Therefore, also $determines_M(T', V', \psi \wedge \gamma, \nu)$ holds for all well-formed assignments ν. Indeed, assume that this is not the case. Then, there are three database states db, db_1, and db_2 such that $db_1, db_2 \in [\![db]\!]_{V',T'}$ and $[(\psi \wedge \gamma) \circ \nu]^{db_1} \neq [(\psi \wedge \gamma) \circ \nu]^{db_2}$. From this and the RC semantics, there are two cases:
   (a) $[\psi \circ \nu]^{db_1} \neq [\psi \circ \nu]^{db_2}$. From this, it follows that $determines_M(T', V', \psi, \nu)$ does not hold. This contradicts the fact that $determines_M(T', V', \psi, \nu)$ holds.
   (b) $[\gamma \circ \nu]^{db_1} \neq [\gamma \circ \nu]^{db_2}$. The proof of this case is similar to the previous one.
2. there is a view v' in $extend(M, s, V')$ such that $def(v') = \{\overline{x} | \psi \wedge \gamma\}$. From Lemma E.1, it follows that there is a view $v'' \in V'$ that is equivalent to v', and, therefore, to $\{\overline{x} | \psi \wedge \gamma\}$. Thus, $determines_M(T', V', \psi \wedge \gamma, \nu)$ holds for all assignments ν that are well-formed for ϕ.

$\phi := \psi \vee \gamma$ This case is similar to $\psi \wedge \gamma$.

$\phi := \neg\psi$ Assume that $apprDet(T', V', \neg\psi, s, M) = \top$. There are two cases:

1. $apprDet(T', V', \psi, s, M) = \top$. From the induction hypothesis, it follows that $determines_M(T', V', \psi, \nu)$ holds. Therefore, also $determines_M(T', V', \neg\psi, \nu)$ holds. Indeed, assume that this is not the case. This means that there are three database states db, db_1, and db_2 such that db_1, db_2 are in $[\![db]\!]_{V',T'}$ and $[\neg\psi \circ \nu]^{db_1} \neq [\neg\psi \circ \nu]^{db_2}$. From this and the RC semantics, it follows that $[\psi \circ \nu]^{db_1} \neq [\psi \circ \nu]^{db_2}$. From this, it follows that $determines_M(T', V', \psi, \nu)$ does not hold. This contradicts the fact that $determines_M(T', V', \psi, \nu)$ holds.
2. there is a view v' in $extend(M, s, V')$ such that $def(v') = \{\overline{x} | \neg\psi\}$. From Lemma E.1, it follows that there is a view $v'' \in V'$ that is equivalent to v', and, therefore, to $\{\overline{x} | \neg\psi\}$. Thus, $determines_M$

$(T', V', \neg\psi, \nu)$ holds for all well-formed assignments ν.

$\boldsymbol{\phi := \exists x.\, \psi}$ Assume that $apprDet(T', V', \exists x.\, \psi, s, M) = \top$. There are two cases:

1. $apprDet(T', V', \psi, s, M) = \top$. From the induction hypothesis, it follows that $determines_M(T', V', \psi, \nu)$ holds for all well-formed assignments ν. Therefore, also $determines_M(T', V', \exists x.\, \psi, \nu)$ holds for all well-formed assignments ν (note that any well-formed assignment for ψ is also a well-formed assignment for $\exists x.\, \psi$). Indeed, assume that this is not the case. This means that there are three database states db, db_1, and db_2 such that db_1, db_2 are in $[\![db]\!]_{V',T'}$ and $[(\exists x.\, \psi) \circ \nu]^{db_1} \neq [(\exists x.\, \psi) \circ \nu]^{db_2}$. From this and the RC semantics, it follows that there is a value $v \in \mathbf{dom}$ such that $[\psi \circ \nu[x \mapsto v]]^{db_1} \neq [\psi \circ \nu[x \mapsto v]]^{db_2}$. Note that $\nu[x \mapsto v]$ is a well-formed assignment for ψ. Let's call the assignment ν'. From this, it follows that $[\psi \circ \nu']^{db_1} \neq [\psi \circ \nu']^{db_2}$. From this, it follows that $determines_M(T', V', \psi, \nu')$ does not hold. This contradicts the fact that $determines_M(T', V', \psi, \nu)$ holds for any well-formed assignment ν.
2. there is a view v' in $extend(M, s, V')$ such that $def(v') = \{\overline{x} | \exists x.\, \psi\}$. From Lemma $E.1$, it follows that there is a view $v'' \in V'$ that is equivalent to v', and, therefore, to $\{\overline{x} | \exists x.\, \psi\}$. Thus, $determines_M(T', V', \exists x.\, \psi, \nu)$ holds for all well-formed assignments for $\exists x.\, \psi$.

$\boldsymbol{\phi := \forall x.\, \psi}$ This case is similar to $\exists x.\, \psi$.

This concludes the proof of the induction step.

This completes the proof. $\square$

We now show that $\rightsquigarrow_{auth}^{appr}$ is a sound approximation of $\rightsquigarrow_{auth}$, i.e., if $s \rightsquigarrow_{auth}^{appr} act$, then $s \rightsquigarrow_{auth} act$. A *derivation of* $s \rightsquigarrow_{auth}^{appr} act$ is a proof tree, obtained using the rules defining $\rightsquigarrow_{auth}^{appr}$, which ends in $s \rightsquigarrow_{auth}^{appr} act$. The size of a derivation is the number of $\rightsquigarrow_{auth}^{appr}$ rules that are used to show that $s \rightsquigarrow_{auth}^{appr} act$. In the following, we switch freely between statements of the form $s \rightsquigarrow_{auth}^{appr} act$ and their derivations. We denote the size of the derivation of $s \rightsquigarrow_{auth}^{appr} act$ as $|s \rightsquigarrow_{auth}^{appr} act|$.

Lemma E.3. *Let $M = \langle D, \Gamma \rangle$ be a system configuration, s be an M-state, c be an M-context, and $act \in \mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D$. If $s \rightsquigarrow_{auth}^{appr} act$, then $s \rightsquigarrow_{auth} act$.*

Proof. Let $M = \langle D, \Gamma \rangle$ be a system configuration, s be an M-state, c be an M-context, and $act \in \mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D$. Furthermore, we assume that there is a derivation of $s \rightsquigarrow_{auth}^{appr} act$. We prove our claim by structural induction on the size of $s \rightsquigarrow_{auth}^{appr} act$'s derivation.

Base Case: We now show that, for all s and act such that $|s \rightsquigarrow_{auth}^{appr} act| = 1$, if $s \rightsquigarrow_{auth}^{appr} act$, then $s \rightsquigarrow_{auth} act$. There are several cases:

1. Rule *INSERT DELETE admin*: If $s \rightsquigarrow_{auth}^{appr} act$, then $s \rightsquigarrow_{auth} act$ follows trivially from the rule's definition.
2. Rule *CREATE VIEW admin*: If $s \rightsquigarrow_{auth}^{appr} act$, then $s \rightsquigarrow_{auth} act$ follows trivially from the rule's definition.
3. Rule *CREATE TRIGGER admin*: If $s \rightsquigarrow_{auth}^{appr} act$, then $s \rightsquigarrow_{auth} act$ follows trivially from the rule's definition.
4. Rule *SELECT*: If $s \rightsquigarrow_{auth}^{appr} act$, then $s \rightsquigarrow_{auth} act$ follows trivially from the rule's definition.
5. Rule *EXECUTE TRIGGER-3*: If $s \rightsquigarrow_{auth}^{appr} act$, then $s \rightsquigarrow_{auth} act$ follows trivially from the rule's definition.
6. Rule *GRANT-2*: If $s \rightsquigarrow_{auth}^{appr} act$, then $s \rightsquigarrow_{auth} act$ follows trivially from the rule's definition.
7. Rule *GRANT-5*: If $s \rightsquigarrow_{auth}^{appr} act$, then $s \rightsquigarrow_{auth} act$ follows trivially from the rule's definition.
8. Rule *ADD USER*: If $s \rightsquigarrow_{auth}^{appr} act$, then $s \rightsquigarrow_{auth} act$ follows trivially from the rule's definition.

Induction Step: We now assume that, for all derivations of size less than $|s \rightsquigarrow_{auth}^{appr} act|$, it holds that if $s' \rightsquigarrow_{auth}^{appr} act'$, then $s' \rightsquigarrow_{auth} act'$. There are several cases:

1. Rule *INSERT DELETE*: Assume that $s \rightsquigarrow_{auth}^{appr} act$ holds and that $act = \langle u, \mathtt{op}', R, \overline{t} \rangle$, where op' is one of $\{\mathtt{INSERT}, \mathtt{DELETE}\}$. From the rule's definition, it follows that there is a grant $g = \langle op, u, \langle op', R \rangle, u' \rangle$ in $s.sec$ such that $s \rightsquigarrow_{auth}^{appr} g$. From this and the induction hypothesis, it follows that $s \rightsquigarrow_{auth} g$. Therefore, $s \rightsquigarrow_{auth} act$ holds because we can apply the *INSERT DELETE* rule in $\rightsquigarrow_{auth}$.
2. Rule *CREATE VIEW*: The proof is similar to the one for the *INSERT DELETE* rule.
3. Rule *CREATE TRIGGER*: The proof is similar to the one for the *INSERT DELETE* rule.
4. Rule *EXECUTE TRIGGER-2*: Assume that $s \rightsquigarrow_{auth}^{appr} act$ holds and that $act = \langle i, o, e, R, \phi, st, A \rangle$ such that $[\phi[\overline{x}^{|R|} \mapsto tpl(s)]]^{s.db} = \top$. From the rule's definition, it follows that both $s \rightsquigarrow_{auth}^{appr} getAction(st, ow, tpl(s))$ and $s \rightsquigarrow_{auth}^{appr} getAction(st, invoker(s), tpl(s))$ hold. From this and the induction hypothesis, both $s \rightsquigarrow_{auth} getAction(st, ow, tpl(s))$ and $s \rightsquigarrow_{auth} getAction(st, invoker(s), tpl(s))$ hold. From this and the *EXECUTE TRIGGER-2* rule in $\rightsquigarrow_{auth}$, it follows that also $s \rightsquigarrow_{auth} act$ holds.
5. Rule *EXECUTE TRIGGER-1*: The proof is similar to the one for the *EXECUTE TRIGGER-2* rule.
6. Rule *GRANT-1*: Assume that $s \rightsquigarrow_{auth}^{appr} act$ holds and that $act = \langle op, u, p, u' \rangle$, where $op \in \{\oplus, \oplus^*\}$. From the rule's definition, it follows that there is a grant $g = \langle \oplus^*, u', p, u'' \rangle$ in $s.sec$ such that $s \rightsquigarrow_{auth}^{appr} g$. From this and the induction hypothesis, ti follows that $s \rightsquigarrow_{auth} g$. From this and the *GRANT-1* rule in $\rightsquigarrow_{auth}$, it follows that $s \rightsquigarrow_{auth} act$ holds.
7. Rule *GRANT-3*: Assume that $s \rightsquigarrow_{auth}^{appr} act$ holds and that $act = \langle op, u, p, o \rangle$, where $p = \langle \mathtt{SELECT}, v \rangle$, $v \in \mathcal{VIEW}_D^{owner}$, $op \in \{\oplus, \oplus^*\}$, and $o = owner(v)$ such that $o \neq admin$. Let T' be the set obtained through the aT function and V' be the set obtained through the aV function. From the rule's definition, it follows that $apprDet(T', V', def(v)) = \top$. From this and Lemma $E.2$, it follows that $determines_M(T', V', def(v))$ holds. We now show that for any $obj \in T' \cup V'$, $hasAccess(s', \{obj\}, o, \oplus^*)$ holds. There are four cases:
   (a) $o = admin$ *and* $obj \in D$. Since $obj \in T'$, it follows that there is a $g = \langle \oplus^*, o, \langle \mathtt{SELECT}, obj \rangle, u' \rangle$ such that $s \rightsquigarrow_{auth}^{appr} g$. From this and the induction hypothesis, it follows that $s \rightsquigarrow_{auth} g$. Therefore, $hasAccess(s', \{obj\}, o, \oplus^*)$ holds.
   (b) $o \neq admin$ *and* $obj \in D$. Since $obj \in T'$, it follows that there is a $g = \langle \oplus^*, o, \langle \mathtt{SELECT}, obj \rangle, u' \rangle$ in sec such that $s \rightsquigarrow_{auth}^{appr} g$. From this and the induction hypothesis, it follows that $s \rightsquigarrow_{auth} g$. Thus, $hasAccess(s', \{obj\}, o, \oplus^*)$ holds.
   (c) $o = admin$ *and* $obj \in V$. The proof of this case is similar to that of $o = admin$ and $obj \in D$.
   (d) $o \neq admin$ *and* $obj \in V$. The proof of this case is similar to that of $o \neq admin$ and $obj \in D$.

Note that from $hasAccess(s', A, o, op)$ and $hasAccess(s', B, o, op)$, it follows that $hasAccess(s', A \cup B, o, op)$. Thus, $hasAccess(s, T' \cup V', o, \oplus^*)$ holds. From this, it follows that $s \rightsquigarrow_{auth} act$ holds because we can apply the corresponding rule in $\rightsquigarrow_{auth}$.
8. Rule *GRANT-4*: The proof is similar to the one for the *GRANT-3* rule.
9. Rule *REVOKE*: Assume that $s \rightsquigarrow_{auth}^{appr} act$ holds and that $act = \langle \ominus, u, p, u' \rangle$. From the rule's definition, it follows that $s' \rightsquigarrow_{auth}^{appr} g$ for any $g \in s'.sec$, where $s' = applyRev(s, \langle \ominus, u, p, u' \rangle)$. From the induction's hypothesis, it follows that $s' \rightsquigarrow_{auth} g$ for any $g \in s'.sec$. Therefore, we can apply the rule *REVOKE* of $\rightsquigarrow_{auth}$ to derive $s \rightsquigarrow_{auth} act$.

This completes our proof. $\square$

E.3 Database Integrity Proofs

We are now ready to prove that f_{int} satisfies the database integrity property.

LEMMA E.4. *For any two states $s = \langle db, U, sec, T, V, c \rangle$, $s' = \langle db', U, sec, T, V, c' \rangle$ in Ω_M and any action $a \in \mathcal{A}_{D,\mathcal{U}}$:*
1. *$s \rightsquigarrow_{auth} a$ iff $s' \rightsquigarrow_{auth} a$, and*
2. *$s \rightsquigarrow_{auth}^{appr} a$ iff $s' \rightsquigarrow_{auth}^{appr} a$.*

PROOF. It is easy to see that the only rules that depends on db, db', c, and c' are `EXECUTE TRIGGER - 1`, `EXECUTE TRIGGER - 2`, and `EXECUTE TRIGGER - 3`. Since they are not used to evaluate whether $s \rightsquigarrow_{auth} a$ and $s \rightsquigarrow_{auth}^{appr} a$ hold for actions in $\mathcal{A}_{D,\mathcal{U}}$, the lemma follows trivially. $\square$

LEMMA E.5. *Let $P = \langle M, f_{int} \rangle$ be an extended configuration, where M is a system configuration, and L be the P-LTS. Then, for all M-states $s = \langle db, U, sec, T, V, c \rangle \in \Omega_M$ such that $trigger(s) = \epsilon$ and all actions $act \in \mathcal{A}_{D,\mathcal{U}}$, if $f_{int}(s, act) = \top$, then $s \rightsquigarrow_{auth} act$.*

PROOF. We prove the theorem by contradiction. Assume, for contradiction's sake, that the claim does not hold. Therefore, there is a state s and an action act such that $f_{int}(s, act) = \top$, $trigger(s) = \epsilon$, and $s \not\rightsquigarrow_{auth} act$. Thus, from $f_{int}(s, act) = \top$, $trigger(s) = \epsilon$, and f_{int}'s definition, it follows $s \rightsquigarrow_{auth}^{appr} act$. From this and Lemma $E.3$, it follows that $s \rightsquigarrow_{auth} act$ leading to a contradiction. $\square$

LEMMA E.6. *Let $P = \langle M, f_{int} \rangle$ be an extended configuration, where M is a system configuration, and L be the P-LTS. Then, for all M-states $s = \langle db, U, sec, T, V, c \rangle \in \Omega_M$ such that $trigger(s) = \epsilon$ and all actions $act \in \mathcal{A}_{D,\mathcal{U}}$, and all M-states s' reachable from s in one step through t, if $secEx(s') = \bot$, then $s \rightsquigarrow_{auth} act$.*

PROOF. We prove the theorem by contradiction. Assume, for contradiction's sake, that the claim does not hold. Therefore, there are two states s and s' and an action act such that s' is reachable in one step from s through act, $secEx(s') = \bot$, $trigger(s) = \epsilon$, and $s \not\rightsquigarrow_{auth} act$. From $secEx(s') = \bot$ and the LTS's rules, it follows that $f_{int}(s, act) = \top$. From this, $trigger(s) = \epsilon$, and f_{int}'s definition, it follows $s \rightsquigarrow_{auth}^{appr} act$. From this and Lemma $E.3$, it follows that $s \rightsquigarrow_{auth} act$ leading to a contradiction. $\square$

LEMMA E.7. *Let $P = \langle M, f_{int} \rangle$ be an extended configuration, where M is a system configuration, and L be the P-LTS. Then, for all M-states $s = \langle db, U, sec, T, V, c \rangle \in \Omega_M$ and all triggers $t \in \mathcal{TRIGGER}_D$ such that $trigger(s) = t$, the following hold:*
1. *If $f_{int}(s, c) = \top$ and $[\psi]^{db} = \bot$, then $s \rightsquigarrow_{auth} t$, where $c = trigCond(s) = \langle u, \texttt{SELECT}, \psi \rangle$.*
2. *If $f_{int}(s, c) = \top$, $[\psi]^{db} = \top$, and $f_{int}(s, a) = \top$, then $s \rightsquigarrow_{auth} t$, where $c = trigCond(s) = \langle u, \texttt{SELECT}, \psi \rangle$ and $a = trigAct(s)$.*

PROOF. We prove both claims by contradiction. Assume, for contradiction's sake, that the first claim does not hold. Therefore, there is a state s and a trigger t such that $f_{int}(s, c) = \top$ and $[\psi]^{db} = \bot$ and $s \not\rightsquigarrow_{auth} t$. From $[\psi]^{db} = \bot$, $trigger(s) = t$, and the rule `EXECUTE TRIGGER - 3`, it follows that $s \rightsquigarrow_{auth} t$ holds, which leads to a contradiction. Assume, for contradiction's sake, that the second claim does not hold. Therefore, there is a state s and a trigger t such that $f_{int}(s, c) = \top$, $[\psi]^{db} = \top$, $f_{int}(s, a) = \top$, and $s \not\rightsquigarrow_{auth} t$. From $f_{int}(s, a) = \top$, it follows $s \rightsquigarrow_{auth}^{appr} t$. From this and Lemma $E.3$, it follows that $s \rightsquigarrow_{auth} t$ holds leading to a contradiction. This completes the proof. $\square$

LEMMA E.8. *Let $P = \langle M, f_{int} \rangle$ be an extended configuration, where M is a system configuration, and L be the P-LTS. Then, for all M-states $s = \langle db, U, sec, T, V, c \rangle \in \Omega_M$, all triggers $t \in \mathcal{TRIGGER}_D$, and all M-states s' reachable from s in one step through t, if $secEx(s') = \bot$, then $s \rightsquigarrow_{auth} t$.*

PROOF. We prove the theorem by contradiction. Assume, for contradiction's sake, that the claim does not hold. Therefore, there are two states s and s' and a trigger t such that s' is reachable in one step through t from s, $secEx(s') = \bot$, and $s \not\rightsquigarrow_{auth} t$. In the following, let $c = \langle u, \texttt{SELECT}, \psi \rangle$ be $trigCond(s)$ and a be $trigAct(s)$. Since t is a trigger, s' is reachable in one-step from s through t, and $secEx(s') = \bot$, there are two cases, according to the LTS rules:
1. $f_{int}(s, c) = \top$ and $[\psi]^{db} = \bot$. In this case, we can always apply the rule `EXECUTE TRIGGER - 3` in the state s to derive $s \rightsquigarrow_{auth} t$ leading to a contradiction.
2. $f_{int}(s, c) = \top$, $[\psi]^{db} = \top$, and $f_{int}(s'', a) = \top$, where s'' is the state obtained from s by updating the context according to the LTS rules. From f_{int}'s definition and $f_{int}(s'', a) = \top$, it follows $s'' \rightsquigarrow_{auth}^{appr} t$. From this and Lemma $E.3$, it follows $s'' \rightsquigarrow_{auth} t$. Since s and s'' are equivalent modulo the context's history and the context's history is not used in the rules defining $\rightsquigarrow_{auth}$, it also follows that $s \rightsquigarrow_{auth} t$ holds. This lead to a contradiction.

Both cases lead to a contradiction. This completes the proof. $\square$

We are now ready to prove our main result, namely that f_{int} provides database integrity.

THEOREM E.1. *Let $P = \langle M, f_{int} \rangle$ be an extended configuration, where $M = \langle D, \Gamma \rangle$ is a system configuration. The PDP f_{int} provides database integrity with respect to P.*

PROOF. To show that f_{int} satisfies the database integrity property, we have to prove that for all reachable states $s = \langle db, U, sec, T, V, c \rangle$:
1. for all states s' reachable from s in one step through an action $a \in \mathcal{A}_{D,\mathcal{U}}$, if $secEx(s') = \bot$, then $s \rightsquigarrow_{auth} a$,
2. for all states s' reachable from s in one step through a trigger $t \in \mathcal{TRIGGER}_D$, if $secEx(s') = \bot$, then $s \rightsquigarrow_{auth} t$.

The first condition has been proved in Lemma $E.6$ and the second one has been proved in Lemma $E.8$. Therefore, f_{int} satisfies the database integrity property. $\square$

We also prove that, by using f_{int}, any reachable state has a consistent policy. This is the underlying reason why f_{int} prevents Attacks 2 and 3.

LEMMA E.9. *Let $P = \langle M, f_{int}\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration. For each reachable state $s = \langle db, U, sec, T, V, c\rangle$, $s \rightsquigarrow_{auth} g$ for all $g \in sec$.*

PROOF. We claim that, for any run r, the state $last(r)$ is such that for all $p \in last(r).sec$, $last(r) \rightsquigarrow_{auth} p$. From this, the lemma follows trivially.

We now prove that for any run r, the state $last(r)$ is such that for all $p \in last(r).sec$, $last(r) \rightsquigarrow_{auth} p$. We do this by structural induction on the length of the run r.

Base Case: The base case consists of the runs containing only one initial state. Note that an initial state contains only grants issued by *admin*, together with views and triggers owned by *admin*. It is easy to see that for any permission $p = \langle op, u, pr, admin\rangle$ in a policy *sec* in an initial state s, it holds that $s \rightsquigarrow_{auth} p$. There are two cases:

1. The privilege pr in p is such that $pr \in \mathcal{PRIV}_D \setminus \mathcal{PRIV}_D^{\texttt{SELECT},\mathcal{VIEW}_D^{owner}}$. Then, $s \rightsquigarrow_{auth} p$ by the rule *GRANT-2*.
2. The privilege pr in p is such that pr is in the set $\mathcal{PRIV}_D^{\texttt{SELECT},\mathcal{VIEW}_D^{owner}}$. Recall that *admin* is the owner of all views in the state. Then, $s \rightsquigarrow_{auth} p$ by the rule *GRANT-3*. Indeed, *admin* can read (and delegate the SELECT permission over) all tables in the database. Therefore, $hasAccess(s, D, admin, \oplus^*)$ and $determines_M(D, \emptyset, q)$ hold for any query q.

This complete the proof for the base case.

Induction Step: We now assume that for all runs r' of length less than the length of r, the state $last(r')$ is such that for all $p \in last(r').sec$, $last(r') \rightsquigarrow_{auth} p$. Let r' be the run $r^{|r|-1}$. There are two cases, depending on whether *act* raises an exception or not.

1. $secEx(last(r)) = \bot$ and $Ex(last(r)) = \emptyset$. There are a number of cases depending on *act*:
   (a) *act* is $\langle u, \texttt{INSERT}, R, \bar{t}\rangle$, $\langle u, \texttt{DELETE}, R, \bar{t}\rangle$, $\langle u, \texttt{SELECT}, q\rangle$, $\langle u, \texttt{ADD\_USER}, u'\rangle$, or $\langle u, \texttt{CREATE}, o\rangle$. In these cases, $last(r').sec = last(r).sec$. Furthermore, $last(r').U \subseteq last(r).U$, $last(r').T \subseteq last(r).T$, and $last(r').V \subseteq last(r).V$. From this and the fact that $last(r') \rightsquigarrow_{auth} g$ for all $g \in last(r').sec$, it follows that $last(r) \rightsquigarrow_{auth} g$ for all $g \in last(r).sec$.
   (b) *act* is $\langle op, u, p, u'\rangle$, where $op \in \{\oplus, \oplus^*\}$. From $secEx(last(r)) = \bot$, it follows that $last(r') \rightsquigarrow_{auth} act$. From the induction hypothesis, it follows that $last(r') \rightsquigarrow_{auth} g$ for all $g \in last(r').sec$. We claim that, for any grant statement g, if $\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} g$, then $\langle db', U, sec', T, V, c'\rangle \rightsquigarrow_{auth} g$ for any policy such that $sec \subseteq sec'$. From the claim, it follows that $last(r) \rightsquigarrow_{auth} act$ and $last(r) \rightsquigarrow_{auth} g$ for all $g \in last(r').sec$. From this and $last(r).sec = \{act\} \cup last(r').sec$, it follows that $last(r) \rightsquigarrow_{auth} g$ for all $g \in last(r).sec$.
   Our claim that, for any grant statement g, if $\langle db, U, sec, T, V, c\rangle \rightsquigarrow_{auth} g$, then $\langle db', U, sec', T, V, c'\rangle \rightsquigarrow_{auth} g$, where $sec \subseteq sec'$, follows trivially from the definition of the rules for GRANT statements.
   (c) *act* is $\langle \ominus, u, p, u'\rangle$. From $secEx(last(r)) = \bot$, it follows that $last(r') \rightsquigarrow_{auth} act$. From this, it follows that $s' \rightsquigarrow^{appr}_{auth} g$ for all $g \in s'.sec$, where $s' = applyRev(last(r'), act)$. From this and Lemma $E.3$, it follows that $s' \rightsquigarrow_{auth} g$ for all $g \in s'.sec$. Recall that $last(r)$ and s' are equivalent modulo the database and the context. From this, Lemma $E.4$, and $s' \rightsquigarrow_{auth} g$ for all $g \in s'.sec$, it follows that $last(r) \rightsquigarrow_{auth} g$ for all $g \in last(r).sec$.
   (d) *act* is a trigger and the WHEN condition is not satisfied. In this case, $last(r')$ and $last(r)$ are equivalent modulo the context. From this, the induction hypothesis, and Lemma $E.4$, it follows that $last(r) \rightsquigarrow_{auth} g$ for all $g \in last(r).sec$.
   (e) *act* is a trigger and the WHEN condition is satisfied. In this case, the proof is the same as the previous cases depending on the trigger's action.
2. $secEx(last(r)) = \top$ or $Ex(last(r)) \neq \emptyset$. From this and the LTS's rules, it follows that there is a state $s' \in \{last(r^i) | 1 \leq i \leq |r| - 1\}$ such that $pState(last(r)) = pState(s')$ (because there has been a roll-back). Let *sec* be the policy in s'. From the induction hypothesis, it follows that for all $p \in sec$, $s' \rightsquigarrow_{auth} p$. From this fact, the $\rightsquigarrow_{auth}$'s definition, $pState(last(r)) = pState(s')$, and Lemma $E.4$, it follows that for all $p \in last(r).sec$, also $last(r) \rightsquigarrow_{auth} p$.

This complete the proof for the induction step.

This completes the proof. $\square$

E.4 Complexity Proofs

THEOREM E.2. *The data complexity of f_{int} is $O(1)$.*

PROOF. Let $M = \langle D, \Gamma\rangle$ be some fixed system configuration, $a \in \mathcal{A}_{D,U}$ be some fixed action, $u \in \mathcal{U}$ be some fixed user, $U \subseteq \mathcal{U}$ be some fixed set of users, $sec \in \Omega^{sec}_{U,D}$ be some fixed policy, T be some fixed set of triggers over D whose owners are in U, V be some fixed set of views over D whose owners are in U, and c be some fixed context. Furthermore, let $db \in \Omega^{\Gamma}_D$ be a database state such that $\langle db, U, sec, T, V, c\rangle \in \Omega_M$. We denote by s the state $\langle db, U, sec, T, V, c\rangle$. We can check whether $f_{int}(s, a) = \top$ as follows:

1. If $trigger(s) = \epsilon$, then return $\top$ iff $s \rightsquigarrow^{appr}_{auth} a$.
2. If $trigger(s) \neq \epsilon$ and $a = trigCond(s)$, return $\top$.
3. If $trigger(s) \neq \epsilon$ and $a = trigAct(s)$, return $\top$ iff both $s \rightsquigarrow^{appr}_{auth} getAction(stmt, ow, tpl(s)) = \top$ and $s \rightsquigarrow^{appr}_{auth} getAction(stmt, invoker(s), tpl(s)) = \top$, where $trigger(s) = \langle id, ow, ev, R, \phi, stmt, m\rangle$.
4. Otherwise return $\bot$.

Note that in the above algorithm we use all the rules in $\rightsquigarrow^{appr}_{auth}$ other than *EXECUTE TRIGGER - 1*, *EXECUTE TRIGGER - 2*, and *EXECUTE TRIGGER - 3*. These are the only rules that depend on the database state. Therefore, evaluating any statement of the form $s \rightsquigarrow^{appr}_{auth} a$ can be done in constant time in terms of data complexity. Therefore, all steps 1–4 can be executed in constant time in terms of data complexity. $\square$

LEMMA E.10. *The complexity of apprDet is $O(|\phi|^3 + |\phi| \cdot max^2 \cdot |V|^3)$.*

PROOF. Let $M = \langle D, \Gamma\rangle$ be a system configuration, $T \subseteq D$ be a set of tables, $V \subseteq \mathcal{VIEW}^{owner}_D$ be a set of views over D, ϕ be a formula over D, and s be an M-state. An algorithm that computes $apprDet(T, V, \phi, s, M)$ is as follows:

1. Compute the set $extend(M, s, V)$.

2. Compute the set S of all sub-formulae of ϕ, i.e., $S = \mathit{subF}(\phi)$. Note that $\phi \in \mathit{subF}(\phi)$.
3. Sort by length the set of sub-formulae in such a way that the shortest formula is the first one.
4. Let $S' := \emptyset$.
5. For each sub-formula ψ in the sequence:
   (a) Check whether there is a view $v \in \mathit{extend}(M, s, V)$ such that ψ is v's definition. If this is the case, let $S' = S' \cup \mathit{subF}(\psi)$.
   (b) Make a case distinction on ψ:
      i. If $\psi := R(\overline{x})$ and $R \in T$, $S' = S' \cup \mathit{subF}(\psi)$.
      ii. If $\psi := V(\overline{x})$ and $\langle V, u, q, O\rangle \in V$, $S' = S' \cup \mathit{subF}(\psi)$.
      iii. If $\psi := \alpha \wedge \beta$ and α, $\beta \in S'$, then $S' = S' \cup \mathit{subF}(\psi)$.
      iv. If $\psi := \alpha \vee \beta$ and α, $\beta \in S'$, then $S' = S' \cup \mathit{subF}(\psi)$.
      v. If $\psi := \neg\alpha$ and $\alpha \in S'$, then $S' = S' \cup \mathit{subF}(\psi)$.
      vi. If $\psi := \exists x.\alpha$ and $\alpha \in S'$, then $S' = S' \cup \mathit{subF}(\psi)$.
      vii. If $\psi := \forall x.\alpha$ and $\alpha \in S'$, then $S' = S' \cup \mathit{subF}(\psi)$.
6. $\mathit{apprDet}(T, V, \phi, s, M) = \top$ iff $S = S'$.

We claim that the size of $\mathit{subF}(\phi)$ is $O(|\phi|)$ and that computing $\mathit{subF}(\psi)$ can be done in $O(|\phi|^2)$. Let max be the maximum length of the definitions of the views in V. We also claim that the size of the set $\mathit{extend}(M, s, V)$ is $O(\mathit{max} \cdot |V|^3)$ and that $\mathit{extend}(M, s, V)$ can be computed in $O(|V|^3 \cdot \mathit{max}^2)$. From these claims, it follows that the fifth step can be executed in $O(|S| \cdot ((|\mathit{extend}(M, s, V)| + |S|) + |S|^2)))$. After some simplification, it follows that the fifth step can be executed in $O(|S|^3 + |S| \cdot |\mathit{extend}(M, s, V)|)$. From this, $|S| = O(|\phi|)$, and $|\mathit{extend}(M, s, V)| = O(\mathit{max} \cdot |V|^3)$, it follows that the fifth step can be executed in $O(|\phi|^3 + |\phi| \cdot \mathit{max} \cdot |V|^3)$. Therefore, the overall complexity is $O(|\phi|^3 + |\phi| \cdot \mathit{max}^2 \cdot |V|^3)$.

We now prove our claims about $\mathit{subF}(\phi)$. It is easy to see that the size of $\mathit{subF}(\phi)$ is $O(|\phi|)$. Indeed, we can view the formula ϕ as a tree, where the operators are the internal nodes and the predicates and equalities are the leaves. Then, there is a sub-formula for each sub-tree. From this and from the fact that the number of sub-tree of a tree is linear in the number of nodes, it follows that $|\mathit{subF}(\phi)|$ is $O(|\phi|)$. Note that computing $\mathit{subF}(\psi)$ can be done in $O(|\phi|^2)$.

We now prove our claims about $\mathit{extend}(M, s, V)$. Let max be the maximum length of the definitions of the views in V. For each $v \in V$, computing $\mathit{inline}(v, s)$ can be done in $O(|v| \cdot |V| \cdot \mathit{max})$. Furthermore, since the views' definitions are acyclic, after $|V|$ applications of inline there are no views in the view's definition. Therefore, for each $v \in V$, we can compute all views derivable from v in $O(|v| \cdot |V|^2 \cdot \mathit{max})$. Therefore, we can compute the set $\mathit{extend}(M, s, V)$ in $O(|V|^3 \cdot \mathit{max}^2)$. Therefore, also the size of $\mathit{extend}(M, s, V)$ is less than $O(\mathit{max} \cdot |V|^3)$. $\square$

$$\frac{\begin{array}{c} u, u' \in U \quad R \in D \quad \bar{t} \in \mathbf{dom}^{|R|} \quad g = \langle op, u, \langle op', R\rangle, u'\rangle \quad g \in sec \\ \langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} g \quad op \in \{\oplus, \oplus^*\} \quad op' \in \{\mathtt{INSERT}, \mathtt{DELETE}\} \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle u, op', R, \bar{t}\rangle} \ \begin{array}{l}\mathtt{INSERT}\\ \mathtt{DELETE}\end{array}$$

$$\frac{\begin{array}{c} u, u' \in U \quad v \in \mathcal{VIEW}_D \quad g = \langle op, u, \langle \mathtt{CREATE\ VIEW}\rangle, u'\rangle \\ g \in sec \quad \langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} g \quad op \in \{\oplus, \oplus^*\} \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle u, \mathtt{CREATE}, v\rangle} \ \begin{array}{l}\mathtt{CREATE}\\ \mathtt{VIEW}\end{array}$$

$$\frac{\begin{array}{c} u, u' \in U \quad t = \langle id, ow, ev, R, \phi, stmt, m\rangle \\ t \in \mathcal{TRIGGER}_D \\ g = \langle op, u, \langle \mathtt{CREATETRIGGER}, R\rangle, u'\rangle \\ g \in sec \quad \langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} g \quad op \in \{\oplus, \oplus^*\} \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle u, \mathtt{CREATE}, t\rangle} \ \begin{array}{l}\mathtt{CREATE}\\ \mathtt{TRIGGER}\end{array}$$

$$\frac{R \in D \quad \bar{t} \in \mathbf{dom}^{|R|} \quad op' \in \{\mathtt{INSERT}, \mathtt{DELETE}\}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle admin, op', R, \bar{t}\rangle} \ \begin{array}{l}\mathtt{INSERT}\\ \mathtt{DELETE}\\ \text{admin}\end{array}$$

$$\frac{v \in \mathcal{VIEW}_D}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle admin, \mathtt{CREATE}, v\rangle} \ \begin{array}{l}\mathtt{CREATE}\\ \mathtt{VIEW}\\ \text{admin}\end{array}$$

$$\frac{t \in \mathcal{TRIGGER}_D}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle admin, \mathtt{CREATE}, t\rangle} \ \begin{array}{l}\mathtt{CREATE}\\ \mathtt{TRIGGER}\\ \text{admin}\end{array}$$

$$\frac{\begin{array}{c} u, u' \in U \quad priv \in \mathcal{PRIV}_D \\ s = \langle db, U, sec, T, V, c\rangle \quad s' = \langle db, U, sec', T, V, c\rangle \\ s' = applyRev(s, \langle \ominus, u, p, u'\rangle) \\ \forall g \in sec'.\, s' \rightsquigarrow^{appr}_{auth} g \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle \ominus, u, priv, u'\rangle} \ \mathtt{REVOKE}$$

$$\frac{\begin{array}{c} u, u', u'' \in U \quad op \in \{\oplus, \oplus^*\} \quad priv \in \mathcal{PRIV}_D \\ g = \langle \oplus^*, u', priv, u''\rangle \quad g \in sec \quad \langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} g \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle op, u, priv, u'\rangle} \ \mathtt{GRANT\text{-}1}$$

$$\frac{\begin{array}{c} u \in U \quad op \in \{\oplus, \oplus^*\} \\ priv \in \mathcal{PRIV}_D \setminus \mathcal{PRIV}_D^{\mathtt{SELECT}, \mathcal{VIEW}^{owner_D}} \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle op, u, priv, admin\rangle} \ \mathtt{GRANT\text{-}2}$$

$$\frac{\begin{array}{c} u, owner \in U \quad op \in \{\oplus, \oplus^*\} \quad priv = \langle \mathtt{SELECT}, v\rangle \\ v = \langle id, owner, q, O\rangle \quad v \in V \\ T' = aT(\langle db, U, sec, T, V, c\rangle, \oplus^*, owner) \\ V' = aV(\langle db, U, sec, T, V, c\rangle, \oplus^*, owner) \\ apprDet(T', V', q) = \top \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle op, u, priv, owner\rangle} \ \mathtt{GRANT\text{-}3}$$

$$\frac{\begin{array}{c} u, owner \in U \quad op \in \{\oplus, \oplus^*\} \quad priv = \langle \mathtt{SELECT}, v\rangle \\ v = \langle id, owner, q, O\rangle \quad v \in V \quad owner \neq admin \\ T' = aT(\langle db, U, sec, T, V, c\rangle, \oplus, owner) \\ V' = aV(\langle db, U, sec, T, V, c\rangle, \oplus, owner) \\ apprDet(T', V', q) = \top \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle op, u, priv, admin\rangle} \ \mathtt{GRANT\text{-}4}$$

$$\frac{\begin{array}{c} u, owner \in U \quad op \in \{\oplus, \oplus^*\} \quad v \in V \\ priv = \langle \mathtt{SELECT}, v\rangle \quad v = \langle id, owner, q, A\rangle \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle op, u, priv, owner\rangle} \ \mathtt{GRANT\text{-}5}$$

$$\frac{u \in \mathcal{U} \quad u' = admin}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle u', \mathtt{ADD\_USER}, u\rangle} \ \mathtt{ADD\ USER}$$

$$\frac{\begin{array}{c} t = \langle id, ow, ev, R, \phi, stmt, O\rangle \quad t \in T \\ \langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} getAction(stmt, ow, tpl(c)) \\ [\phi[\overline{x}^{|R|} \mapsto tpl(c)]]^{db} = \top \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} t} \ \begin{array}{l}\mathtt{EXECUTE}\\ \mathtt{TRIGGER\text{-}1}\end{array}$$

$$\frac{\begin{array}{c} t = \langle id, ow, ev, R, \phi, stmt, A\rangle \quad t \in T \\ \langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} getAction(stmt, invoker(c), tpl(c)) \\ \langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} getAction(stmt, ow, tpl(c)) \\ [\phi[\overline{x}^{|R|} \mapsto tpl(c)]]^{db} = \top \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} t} \ \begin{array}{l}\mathtt{EXECUTE}\\ \mathtt{TRIGGER\text{-}2}\end{array}$$

$$\frac{u \in U \quad q \in RC}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle u, \mathtt{SELECT}, q\rangle} \ \mathtt{SELECT}$$

$$\frac{\begin{array}{c} t = \langle id, ow, ev, R, \phi, stmt, m\rangle \\ t \in T \quad [\phi[\overline{x}^{|R|} \mapsto tpl(c)]]^{db} = \bot \end{array}}{\langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} t} \ \begin{array}{l}\mathtt{EXECUTE}\\ \mathtt{TRIGGER\text{-}3}\end{array}$$

$$\begin{aligned} aT(\langle db, U, sec, T, V, c\rangle, op, u) = \{R \in D \mid (u = admin \wedge \exists u' \in U, op' \in \{\oplus^*, op\}.\ & \\ \langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle op', u, \langle \mathtt{SELECT}, R\rangle, u'\rangle) \vee & \\ \exists u' \in U, g \in sec, op' \in \{\oplus^*, op\}.\, g = \langle op', u, \langle \mathtt{SELECT}, R\rangle, u'\rangle & \\ \wedge \langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} g\} & \end{aligned}$$

$$\begin{aligned} aV(\langle db, U, sec, T, V, c\rangle, op, u) = \{V \in V \cap \mathcal{VIEW}_D^{owner} \mid (u = admin \wedge \exists u' \in U, op' \in \{\oplus^*, op\}.\ & \\ \langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} \langle op', u, \langle \mathtt{SELECT}, V\rangle, u'\rangle) \vee & \\ \exists u' \in U, g \in sec, op' \in \{\oplus^*, op\}.\, g = \langle op', u, \langle \mathtt{SELECT}, V\rangle, u'\rangle & \\ \wedge \langle db, U, sec, T, V, c\rangle \rightsquigarrow^{appr}_{auth} g\} & \end{aligned}$$

Figure 37: Definition of the $\rightsquigarrow^{appr}_{auth}$ relation

F. ENFORCING DATA CONFIDENTIALITY

Here, we first formalize the PDP f_{conf}. Afterwards, we prove that it provides the data confidentiality property. Finally, we show that its data complexity is AC^0.

Let $M = \langle D, \Gamma \rangle$ be a system configuration. The PDP f^u_{conf} is shown in Figure 38. The function is parametrized by the user u against which the PDP provides data confidentiality. The PDP $f^u_{conf}(s, a)$ models the function $f_{conf}(s, a, u)$ shown in Figure 8. The mapping between the PDP f^u_{conf} and the pseudo-code shown in Figure 8 is immediate.

The PDP f^u_{conf} uses a number of auxiliary functions. Recall that the function tr, defined in Appendix A, takes as input an M-state $s \in \Omega_M$ and returns the definition of the trigger that the system is executing. If the system is not executing any trigger, then $tr(s) = \epsilon$. Equivalently, $tr(s)$ is the first trigger in the sequence of triggers returned by $triggers(s)$.

The function $tDet$ takes as input a view $v = \langle i, o, \{\overline{x}|\phi\}, m\rangle \in \mathcal{VIEW}_D$, a state $s \in \Omega_M$, and a system configuration $M = \langle D, \Gamma \rangle$ and returns as output the smallest set of tables in D that determines v, namely the smallest set $T \in \mathbb{P}(D)$ such that $apprDet(T, \emptyset, \phi, s, M)$ holds, where $apprDet$ is defined in Appendix E. Note that such a set is always unique.

The function $noLeak$, defined in Figure 38, takes as input a state s, an `INSERT` or `DELETE` action a, and a user u and checks whether the execution of the action a may leak sensitive information through the views that the user u can read, as shown in Example 5.4. Note that the function $noLeak$ returns $\top$ if there is no leakage of sensitive information and returns $\bot$ if the action a may leak sensitive information through the views the user u can read in the state s. We remark that the function $leak(a, s, u)$ used in the algorithm in Section 6 returns is defined as $leak(a, s, u) = noLeak(s, a, u)$.

We now define the Dep, $getInfoS$, $getInfoV$, and $getInfo$ functions. The function Dep is as follows. $Dep(\langle u, \texttt{INSERT}, R, \overline{t}\rangle, \Gamma)$ returns the set containing all the formulae in Γ of the form $\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. (R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}'$ or $\forall \overline{x}, \overline{z}. R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w})$, whereas $Dep(\langle u, \texttt{DELETE}, R, \overline{t}\rangle, \Gamma)$ returns the set containing all the formulae in Γ of the form $\forall \overline{x}, \overline{z}. S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. R(\overline{x}, \overline{w})$.

The function $getInfoS$ is defined as follows:

- $getInfoS(\langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle, \phi^R_{funct})$ is the formula $\neg\exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$, where ϕ^R_{funct} is a formula of the form $\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. (R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}'$.
- $getInfoS(\langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w})\rangle, \phi^{R,S}_{incl})$ is the formula $\exists \overline{y}. S(\overline{v}, \overline{y})$, where $\phi^{R,S}_{incl}$ is a formula of the form $\forall \overline{x}, \overline{z}. R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w})$.
- $getInfoS(\langle u, \texttt{DELETE}, R, (\overline{v}, \overline{w})\rangle, \phi^{S,R}_{incl})$ is the formula $\forall \overline{x}, \overline{z}. (S(\overline{x}, \overline{z}) \Rightarrow \overline{x} \neq \overline{v}) \vee \exists \overline{y}. (R(\overline{v}, \overline{y}) \wedge \overline{y} \neq \overline{w})$, where $\phi^{S,R}_{incl}$ is a formula of the form $\forall \overline{x}, \overline{z}. S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. R(\overline{x}, \overline{w})$.
- $getInfoS(act, \phi) = \top$ otherwise.

The function $getInfoV$ is defined as follows:

- $getInfoV(\langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle, \phi^R_{funct})$ is the formula $\exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$, where ϕ^R_{funct} is a formula of the form $\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. (R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}'$.
- $getInfoV(\langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w})\rangle, \phi^{R,S}_{incl})$ is the formula $\forall \overline{x}, \overline{y}. S(\overline{x}, \overline{y}) \Rightarrow \overline{x} \neq \overline{v}$, where $\phi^{R,S}_{incl}$ is a formula of the form $\forall \overline{x}, \overline{z}. R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w})$.
- $getInfoV(\langle u, \texttt{DELETE}, R, (\overline{v}, \overline{w})\rangle, \phi^{S,R}_{incl})$ is the formula $\exists \overline{z}. S(\overline{v}, \overline{z}) \wedge \forall \overline{y}. (R(\overline{v}, \overline{y}) \Rightarrow \overline{y} = \overline{w})$, where $\phi^{S,R}_{incl}$ is a formula of the form $\forall \overline{x}, \overline{z}. S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. R(\overline{x}, \overline{w})$.
- $getInfoV(act, \phi) = \top$ otherwise.

The function $getInfo$ is as follows:

$$getInfo(\langle u, op, R, \overline{t}\rangle) = \begin{cases} \neg R(\overline{t}) & \text{if } op = \texttt{INSERT} \\ R(\overline{t}) & \text{if } op = \texttt{DELETE} \end{cases}$$

In §F.1 we describe the $secure$ function and we show that it is a sound, under-approximation of the concept of secure judgments. Afterwards, in §F.2 we prove that f^u_{conf} provides data confidentiality with respect to the user u. Finally, in §F.3 we prove that the data complexity of f^u_{conf} is AC^0. In the rest of the paper, instead of writing $secure_{P,\cong_{P,u}}$ we simply write $secure_{P,u}$ and we omit the reference to the indistinguishability relation $\cong_{P,u}$ defined in Appendix D.

F.1 Checking a judgment's security

We still have to define the $secure : \mathcal{U} \times RC_{bool} \times \Omega_M \rightarrow \{\top, \bot\}$ function that determines a given judgment's security. In more detail, the $secure$ function is as follows:

$$secure(u, \phi, s) = \begin{cases} \top & \text{if } [\phi^{rw}_{s,u}]^{s.db} = \bot \\ \bot & \text{otherwise} \end{cases}$$

In the following, we assume that both the formula ϕ and the set of views V in the state s contain just views with owner's privileges. This is without loss of generality. Indeed, views with activator's privileges are just syntactic sugar, they do not disclose additional information to a user other than what he is already authorized to read because they are executed under the activator's privileges. If ϕ and s contain views with activator's privileges, we can compute another formula ϕ' and a state s' without views with activator's privileges as follows. We replace, in the formula ϕ, the predicates of the form $V(\overline{x})$, where V is a view with activator's privileges, with V's definition, and we repeat this process until the resulting formula ϕ' no longer contains views with activator's privileges. Similarly, the set V' is obtained from V by (1) removing all views with activator's privileges, and (2) for each view $v \in V$ with owner's privileges, replacing the predicates of the form $V(\overline{x})$ in v's definition, where V is a view with activator's privileges, with V's definition until v's definition no longer contains views with activator's privileges. The security policy sec' is also obtained from sec by removing all references to views with activator's privileges. Finally, $secure(u, \phi, \langle db, U, sec, T, V, c\rangle)$ is just $secure(u, \phi', \langle db, U, sec', T, V', c\rangle)$.

Before defining the $\phi^{\top}_{s,u}$ and $\phi^{\bot}_{s,u}$ rewritings, we define query containment. Let $M = \langle D, \Gamma \rangle$ be a system configuration. Given two formulae $\phi(\overline{x})$ and $\psi(\overline{y})$, we write $\phi \subseteq_M \psi$ to denote that ϕ is contained in ψ, i.e., $\forall d \in \Omega^{\Gamma}_D. [\{\overline{x}|\phi\}]^d \subseteq [\{\overline{y}|\psi\}]^d$. Determining whether $\phi \subseteq_M \psi$ holds is undecidable for RC [3]. Hence, we develop a sound, under-approximation of query containment. Figure 39 describes the rules defining our under-approximation. For simplicity's sake, the rules are defined only for relational calculus formulae that do not use views. To check whether $\phi \subseteq_M \psi$ holds for two formulae ϕ and ψ that use views, we first compute the formulae ϕ' and ψ', obtained by replacing views' identifiers with their definitions, and then we check whether $\phi' \subseteq_M \psi'$ using the rules in Figure 39. This preserves containment since ϕ and ψ are semantically equivalent to ϕ' and ψ'. Both in the rules and in the proof of Lemma F.1, we assume that there is a total ordering $\preceq_{var}$ over the set of all possible variable identifiers. This ensures that, given a formula ϕ, there is a unique non-boolean query $\{\overline{x} \mid \phi\}$ associated to it, where the

$$f^u_{conf}(s, act) = \begin{cases} f^u_{conf,\mathtt{S}}(s, act) & \text{if } act = \langle u', \mathtt{SELECT}, q\rangle \\ f^u_{conf,\mathtt{I},\mathtt{D}}(s, act) & \text{if } act = \langle u', \mathtt{INSERT}, R, \bar{t}\rangle \\ f^u_{conf,\mathtt{I},\mathtt{D}}(s, act) & \text{if } act = \langle u', \mathtt{DELETE}, R, \bar{t}\rangle \\ f^u_{conf,\mathtt{G},\mathtt{R}}(s, act) & \text{if } act = \langle op, u'', p, u'\rangle \wedge op \in \{\oplus, \oplus^*\} \\ \top & \text{if } u = admin \\ \top & \text{otherwise} \end{cases}$$

$$f^u_{conf,\mathtt{I},\mathtt{D}}(s, act) = \begin{cases} \begin{array}{l} secure(u, getInfo(act), s) \wedge \\ \bigwedge_{\gamma \in Dep(act,\Gamma)} secure(u, getInfoS(\gamma, act), s) \\ \quad \wedge secure(u, getInfoV(\gamma, act), s) \end{array} & \text{if } act = \langle u, op, R, \bar{t}\rangle \wedge trigger(s) = \epsilon \wedge noLeak(s, act, u) = \top \\ \bot & \text{if } act = \langle u, op, R, \bar{t}\rangle \wedge trigger(s) = \epsilon \wedge noLeak(s, act, u) = \bot \\ \begin{array}{l} secure(u, getInfo(act), s) \wedge \\ \bigwedge_{\gamma \in Dep(act,\Gamma)} secure(u, getInfoS(\gamma, act), s) \\ \quad \wedge secure(u, getInfoV(\gamma, act), s) \end{array} & \text{if } invoker(s) = u \wedge trigger(s) \neq \epsilon \wedge noLeak(s, act, u) = \top \\ \bot & \text{if } invoker(s) = u \wedge trigger(s) \neq \epsilon \wedge noLeak(s, act, u) = \bot \\ \top & \text{otherwise} \end{cases}$$

$$f^u_{conf,\mathtt{S}}(s, \langle u', \mathtt{SELECT}, q\rangle) = \begin{cases} secure(u, q, s) & \text{if } u' = u \wedge trigger(s) = \epsilon \\ secure(u, q, s) & \text{if } invoker(s) = u \wedge trigger(s) \neq \epsilon \\ \top & \text{otherwise} \end{cases}$$

$$f^u_{conf,\mathtt{G}}(s, \langle op, u'', p, u'\rangle) = \begin{cases} \bot & \text{if } u'' = u \wedge u' = u \wedge trigger(s) = \epsilon \wedge op \in \{\oplus, \oplus^*\} \wedge p = \langle \mathtt{SELECT}, O\rangle \wedge \\ & \quad \langle \oplus, \mathtt{SELECT}, O\rangle \notin permissions(s, u) \\ \bot & \text{if } u'' = u \wedge invoker(s) = u \wedge trigger(s) \neq \epsilon \wedge op \in \{\oplus, \oplus^*\} \wedge p = \langle \mathtt{SELECT}, O\rangle \wedge \\ & \quad \langle \oplus, \mathtt{SELECT}, O\rangle \notin permissions(s, u) \\ \top & \text{otherwise} \end{cases}$$

$$noLeak(s, \langle u', op, R, \bar{t}\rangle, u) = \begin{cases} \top & \text{if } u' = u \wedge trigger(s) = \epsilon \wedge \forall v \in \mathcal{VIEW}_D. (((\langle \oplus, \mathtt{SELECT}, v\rangle \in permissions(s, u) \wedge \\ & \quad R \in tDet(v, s, M)) \Rightarrow (\forall o \in tDet(v, s, M).\, \langle \oplus, \mathtt{SELECT}, o\rangle \in permissions(s, u))) \\ \top & \text{if } invoker(s) = u \wedge trigger(s) \neq \epsilon \wedge \forall v \in \mathcal{VIEW}_D. (((\langle \oplus, \mathtt{SELECT}, v\rangle \in permissions(s, u) \wedge \\ & \quad R \in tDet(v, s, M)) \Rightarrow (\forall o \in tDet(v, s, M).\, \langle \oplus, \mathtt{SELECT}, o\rangle \in permissions(s, u))) \\ \bot & \text{otherwise} \end{cases}$$

Figure 38: Access control function f^u_{conf}

variables in $\overline{x}$ are those in $\mathit{free}(\phi)$ ordered according to $\preceq_{var}$.

Lemma $F.1$ proves that the rules in Figure 39 are a sound, under-approximation of query containment.

Lemma F.1. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, and $\phi(\overline{x})$ and $\psi(\overline{y})$ be two formulae. If $\phi \subseteq_M \psi$, according to the rules in Figure 39, then $\forall d \in \Omega_D^\Gamma. [\{\overline{x}|\phi\}]^d \subseteq [\{\overline{y}|\psi\}]^d$, where $\overline{x}$ (respectively $\overline{y}$) is the tuple defined by the variables in $\mathit{free}(\phi)$ (respectively $\mathit{free}(\psi)$) ordered according to $\preceq_{var}$.*

Proof. $\phi \subseteq_M \psi$ iff there is a finite derivation that ends in $\phi \subseteq_M \psi$ created using the rules in Figure 39. We prove our claim by structural induction on the derivation's length.

Base Case Assume that the derivation has length 1. There are four cases depending on the rule used to derive $\phi \subseteq_M \psi$:

1. Rule *And*. From the rule's definition, it follows that $\mathit{free}(\phi) = \mathit{free}(\phi \wedge \psi) = \overline{x}$. Let $d \in \Omega_D^\Gamma$ and $\overline{t} \in [\{\overline{x} \mid \phi \wedge \psi\}]^d$. From $\overline{t} \in [\{\overline{x} \mid \phi \wedge \psi\}]^d$ and the definition of non-boolean query, it follows that $[(\phi \wedge \psi)[\overline{x} \mapsto \overline{t}]]^d = \top$. From this and the relational calculus semantics, it follows that $[\phi[\overline{x} \mapsto \overline{t}]]^d = \top$. From this and the definition of non-boolean query, $\overline{t} \in [\{\overline{x} \mid \phi\}]^d$. Therefore, $[\{\overline{x}|\phi \wedge \psi\}]^d \subseteq [\{\overline{x}|\phi\}]^d$.
2. Rule *Or*. From the rule's definition, it follows that $\mathit{free}(\phi) = \mathit{free}(\phi \vee \psi) = \overline{x}$. Let $d \in \Omega_D^\Gamma$ and $\overline{t} \in [\{\overline{x} \mid \phi\}]^d$. From $\overline{t} \in [\{\overline{x} \mid \phi\}]^d$ and the definition of non-boolean query, it follows that $[\phi[\overline{x} \mapsto \overline{t}]]^d = \top$. From this and the relational calculus semantics, it follows that $[(\phi \vee \psi)[\overline{x} \mapsto \overline{t}]]^d = \top$. From this and the definition of non-boolean query, $\overline{t} \in [\{\overline{x} \mid \phi \vee \psi\}]^d$. Therefore, $[\{\overline{x}|\phi\}]^d \subseteq [\{\overline{x}|\phi \vee \psi\}]^d$.
3. Rule *Identity*. From the rule's definition, it follows that $\mathit{free}(\phi) = \overline{x}$, $\mathit{free}(\psi) = \overline{y}$, and $\phi[\overline{x} \mapsto \overline{y}] = \psi$. Let $d \in \Omega_D^\Gamma$ and $\overline{t} \in [\{\overline{x} \mid \phi\}]^d$. From $\overline{t} \in [\{\overline{x} \mid \phi\}]^d$ and the definition of non-boolean query, it follows that $[\phi[\overline{x} \mapsto \overline{t}]]^d = \top$. From this and $\phi[\overline{x} \mapsto \overline{y}] = \psi$, it follows that $[\psi[\overline{y} \mapsto \overline{t}]]^d = \top$. From this and the definition of non-boolean query, $\overline{t} \in [\{\overline{y} \mid \psi\}]^d$. Therefore, $[\{\overline{x}|\phi\}]^d \subseteq [\{\overline{y}|\psi\}]^d$.
4. Rule *Inclusion Dependency*. From the rule's definition, it follows that $\gamma := \forall \overline{x}, \overline{z}. (R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w}))$ is in Γ. Let $d \in \Omega_D^\Gamma$ and $\overline{t} \in [\{\overline{x} \mid \exists \overline{z}. R(\overline{x}, \overline{z})\}]^d$. From $\overline{t} \in [\{\overline{x} \mid \exists \overline{z}. R(\overline{x}, \overline{z})\}]^d$ and the definition of non-boolean query, it follows that $[\exists \overline{z}. R(\overline{t}, \overline{z})]^d = \top$. Therefore, there is a tuple $(\overline{t}, \overline{w}) \in d(R)$. From this and $\gamma \in \Gamma$, it follows that there is a tuple $(\overline{t}, \overline{w}') \in d(S)$. From this, it follows that $[\exists \overline{w}. S(\overline{t}, \overline{w})]^d = \top$. From this and the definition of non-boolean query, it follows that $\overline{t} \in [\{\overline{x} \mid \exists \overline{w}. S(\overline{x}, \overline{w})\}]^d$. Therefore, it follows that $[\{\overline{x} \mid \exists \overline{z}. R(\overline{x}, \overline{z})\}]^d \subseteq [\{\overline{x} \mid \exists \overline{w}. S(\overline{x}, \overline{w})\}]^d$ holds.

This completes the proof for the base case.

Induction Step Assume now that the claim holds for all derivations of length less than that of $\phi \subseteq_M \psi$. We now prove that it holds also for $\phi \subseteq_M \psi$. There is just one case, namely $\phi \subseteq_M \psi$ is of the form $\exists x_i. \alpha \subseteq_M \exists y_i. \beta$ and it is obtained by applying the rule *Projection* to $\alpha \subseteq_M \beta$. From the rule, it follows that $\alpha \subseteq_M \beta$ holds. Let $1 \leq u \leq n$ and $\overline{t}'$ (respectively $\overline{x}'$ and $\overline{y}'$) be the tuple obtained from $\overline{t}$ (respectively $\overline{x}$ and $\overline{y}$) by dropping the i-th value (respectively variable). We now prove that $[\{\overline{x}'|\exists x_i. \alpha\}]^d \subseteq [\{\overline{y}'|\exists y_i. \beta\}]^d$. Assume, for contradiction's sake, that this is not the case, namely there is a tuple $\overline{v}$ such that $\overline{v} \in [\{\overline{x}'|\exists x_i. \alpha\}]^d$ but $\overline{v} \notin [\{\overline{y}'|\exists y_i. \beta\}]^d$. From $\overline{v} \in [\{\overline{x}'|\exists x_i. \alpha\}]^d$ and the relational calculus semantics, it follows that there is a tuple $\overline{v}_1$, obtained by adding a value to $\overline{v}$ in the i-th position, such that $\overline{v}_1 \in [\{\overline{x}|\alpha\}]^d$. From this, $\alpha \subseteq_M \beta$, and the induction hypothesis, it follows that $\overline{v}_1 \in [\{\overline{y}|\beta\}]^d$. From this and the relational calculus semantics, it follows that $\overline{v} \in [\{\overline{y}'|\exists y_i. \beta\}]^d$. This contradicts the fact that $\overline{v} \notin [\{\overline{y}'|\exists y_i. \beta\}]^d$.

This completes the proof. □

$$\frac{\mathit{free}(\phi \wedge \psi) = \mathit{free}(\phi) \quad M = \langle D, \Gamma\rangle \quad \mathit{free}(\phi) \neq \emptyset}{\phi \wedge \psi \subseteq_M \phi}\ \text{And} \qquad \frac{\mathit{free}(\phi) = \mathit{free}(\phi \vee \psi) \quad M = \langle D, \Gamma\rangle \quad \mathit{free}(\phi) \neq \emptyset}{\phi \subseteq_M \phi \vee \psi}\ \text{Or}$$

$$\frac{M = \langle D, \Gamma\rangle \quad n > 1 \quad \mathit{free}(\phi) = \{x_1, \ldots, x_n\} \quad \mathit{free}(\psi) = \{y_1, \ldots, y_n\} \quad 1 \leq i \leq n \quad \phi \subseteq_M \psi}{\exists x_i. \phi \subseteq_M \exists y_i. \phi}\ \text{Projection}$$

$$\frac{M = \langle D, \Gamma\rangle \quad n > 0 \quad \mathit{free}(\phi) = \{x_1, \ldots, x_n\} \quad \mathit{free}(\psi) = \{y_1, \ldots, y_n\} \quad \phi[x_1 \mapsto y_1, \ldots, x_n \mapsto y_n] = \psi}{\phi \subseteq_M \psi}\ \text{Identity}$$

$$\frac{M = \langle D, \Gamma\rangle \quad |\overline{x}| > 0 \quad \forall \overline{x}, \overline{z}. (R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w})) \in \Gamma}{\exists \overline{z}. R(\overline{x}, \overline{z}) \subseteq_M \exists \overline{w}. S(\overline{x}, \overline{w})}\ \text{Inclusion Dependency}$$

Figure 39: Containment rules

Given a table or a view O and a sequence of distinct integers $\overline{i} := (i_1, \ldots, i_n)$ such that $1 \leq i_j \leq |O|$ for all $1 \leq j \leq n$, where $0 \leq n < |O|$, the *$\overline{i}$-projection of* O, denoted by $O_{\overline{i}}$, is the formula $\exists x_{i_1}, \ldots, x_{i_n}. O(x_1, \ldots, x_{|O|})$. Given a database schema D and a set of views V defined over D, we denote by $\mathit{extVocabulary}(D, V)$ the extended vocabulary obtained by defining all possible projections of tables in D and views in V, i.e., for each $O \in D \cup V$, we define a predicate $O_{\overline{i}}$ for each projection $\exists x_{i_1}, \ldots, x_{i_n}. O(x_1, \ldots, x_{|O|})$ of O. Furthermore, given a relational calculus formula ϕ over D, we denote by $\mathit{extVoc}_{V,D}(\phi)$ the formula obtained by replacing all sub-formulae of the form $\exists \overline{x}. R(\overline{x}, \overline{y})$ with the predicates in $\mathit{extVocabulary}(D, V)$ representing the corresponding projections $R_{\overline{i}}$. Finally, we denote by $\mathit{inline}_{D,V}(\phi)$, where ϕ is a relational calculus formula over $\mathit{extVocabulary}(D, V)$, the formula ϕ' obtained by replacing all predicates associated with projections with the corresponding formulae.

Let S be predicate in $\mathit{extVocabulary}(D, V)$ and s be an M-state. We denote by $S_s^\top$ the set of all projections of tables in D and views in V that are contained in S, i.e., $S_s^\top := \{R \in \mathit{extVocabulary}(D, V) \mid R(\overline{x}) \subseteq_M S(\overline{y})\}$[3]. Similarly, we denote by $S_s^\perp$ the set of all projections of tables in D and views in V that contains S, i.e., $S_s^\perp := \{R \in \mathit{extVocabulary}(D, V) \mid S(\overline{x}) \subseteq_M R(\overline{y})\}$. Furthermore, we denote by $AUTH_{s,u}$ the set of all tables and views that u is authorized to read in s, i.e., $AUTH_{s,u} := \{v \mid \langle \oplus, \texttt{SELECT}, v\rangle \in \mathit{permissions}(s, u)\}$, and by $AUTH^*_{s,u}$ the set of all the projections obtained from tables and views in $AUTH_{s,u}$.

We are now ready to formally define the $\phi^\top_{s,u}$ and $\phi^\perp_{s,u}$ rewritings.

Definition F.1. Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V, c\rangle$ be an M-state, u be a user, and ϕ be a relational calculus sentence over $\mathit{extVocabulary}(D, V)$.

The function *bound* takes as input a formula ϕ, a state s, a user u, a variable identifier x, and a value v in $\{\top, \perp\}$. It is inductively defined as follows:

- $\mathit{bound}(R(\overline{y}), s, u, x, v)$, where R is a predicate symbol in $\mathit{extVocabulary}(D, V)$, is $\top$ iff (a) x occurs in $\overline{y}$, and (b) the set $R^v_{s,u}$, which is $R^v_s \cap AUTH^*_{s,u}$, is not empty.

[3]With a slight abuse of notation, we write $R(\overline{x}) \subseteq_M S(\overline{y})$ instead of $\mathit{inline}_{D,V}(R(\overline{x})) \subseteq_M \mathit{inline}_{D,V}(S(\overline{y}))$.

- $bound(y = z, s, u, x, v)$ is $\top$ iff $x = y$ and z is a constant symbol or $x = z$ and y is a constant symbol.
- $bound(\top, s, u, x, v) := \bot$.
- $bound(\bot, s, u, x, v) := \bot$.
- $bound(\neg\psi, s, u, x, v) := bound(\psi, s, u, x, \neg v)$, where ψ is a relational calculus formula.
- $bound(\psi\wedge\gamma, s, u, x, v) := bound(\psi, s, u, x, v) \vee bound(\gamma, s, u, x, v)$, where ψ and γ are relational calculus formulae.
- $bound(\psi\vee\gamma, s, u, x, v) := bound(\psi, s, u, x, v) \wedge bound(\gamma, s, u, x, v)$, where ψ and γ are relational calculus formulae.
- $bound(\exists y.\psi, s, u, x, v)$, where ψ is a relational calculus formula, is $bound(\psi, s, u, x, v) \wedge bound(\psi, s, u, y, v)$ if $x \neq y$, and $bound(\exists y.\psi, s, u, x, v) := \bot$ otherwise.
- $bound(\forall y.\psi, s, u, x, v)$, where ψ is a relational calculus formula, is $bound(\psi, s, u, x, v) \wedge bound(\psi, s, u, y, v)$ if $x \neq y$, and $bound(\forall y.\psi, s, u, x, v) := \bot$ otherwise.

The formula $\phi^{\top}_{s,u}$ is inductively defined as follows:

- $R(\overline{x})^{\top}_{s,u} := \bigvee_{S\in R^{\top}_{s,u}} S(\overline{x})$, where R is a predicate symbol in $extVocabulary(D, V)$ and $R^{\top}_{s,u} := R^{\top}_{s} \cap AUTH^{*}_{s,u}$.
- $(x = v)^{\top}_{s,u} := (x = v)$, where x and v are either variable identifiers or constant symbols.
- $(\top)^{\top}_{s,u} := \top$.
- $(\bot)^{\top}_{s,u} := \bot$.
- $(\neg\psi)^{\top}_{s,u} := \neg\psi^{\bot}_{s,u}$, where ψ is a relational calculus formula.
- $(\psi \wedge \gamma)^{\top}_{s,u} := \psi^{\top}_{s,u} \wedge \gamma^{\top}_{s,u}$, where ψ and γ are relational calculus formulae.
- $(\psi \vee \gamma)^{\top}_{s,u} := \psi^{\top}_{s,u} \vee \gamma^{\top}_{s,u}$, where ψ and γ are relational calculus formulae.
- $(\exists x.\, \psi)^{\top}_{s,u}$, where ψ is a relational calculus formula and x is a variable identifier, is $\exists x.\, \psi^{\top}_{s,u}$ if $bound(\psi, s, u, x, \top) = \top$ and $(\exists x.\, \psi)^{\top}_{s,u} := \bot$ otherwise.
- $(\forall x.\, \psi)^{\top}_{s,u}$, where ψ is a relational calculus formula and x is a variable identifier, is $\forall x.\, \psi^{\top}_{s,u}$ if $bound(\psi, s, u, x, \top) = \top$ and $(\forall x.\, \psi)^{\top}_{s,u} := \bot$ otherwise.

The formula $\phi^{\bot}_{s,u}$ is inductively defined as follows:

- $R(\overline{x})^{\bot}_{s,u} := \bigwedge_{S\in R^{\bot}_{s,u}} S(\overline{x})$, where R is a predicate symbol in $extVocabulary(D, V)$ and $R^{\bot}_{s,u} := R^{\bot}_{s} \cap AUTH^{*}_{s,u}$.
- $(x = v)^{\bot}_{s,u} := (x = v)$, where x and v are either variable identifiers or constant symbols.
- $(\top)^{\bot}_{s,u} := \top$.
- $(\bot)^{\bot}_{s,u} := \bot$.
- $(\neg\psi)^{\bot}_{s,u} := \neg\psi^{\top}_{s,u}$, where ψ is a relational calculus formula.
- $(\psi \wedge \gamma)^{\bot}_{s,u} := \psi^{\bot}_{s,u} \wedge \gamma^{\bot}_{s,u}$, where ψ and γ are relational calculus formulae.
- $(\psi \vee \gamma)^{\bot}_{s,u} := \psi^{\bot}_{s,u} \vee \gamma^{\bot}_{s,u}$, where ψ and γ are relational calculus formulae.
- $(\exists x.\, \psi)^{\bot}_{s,u}$, where ψ is a relational calculus formula and x is a variable identifier, is $\exists x.\, \psi^{\bot}_{s,u}$ if $bound(\psi, s, u, x, \bot) = \top$ and $(\exists x.\, \psi)^{\bot}_{s,u} := \bot$ otherwise.
- $(\forall x.\, \psi)^{\bot}_{s,u}$, where ψ is a relational calculus formula and x is a variable identifier, is $\forall x.\, \psi^{\bot}_{s,u}$ if $bound(\psi, s, u, x, \bot) = \top$ and $(\forall x.\, \psi)^{\bot}_{s,u} := \bot$ otherwise. $\square$

Finally, we define the formula $\phi^{rw}_{s,u}$ which represents the overall rewritten formula.

Definition F.2. Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V, c\rangle$ be an M-state, u be a user, and ϕ be a relational calculus sentence over D. The formula $\phi^{rw}_{s,u}$ is defined as $inline_{V,D}(\neg\psi^{\top}_{s,u} \wedge \psi^{\bot}_{s,u})$, where $\psi := extVoc_{V,D}(\phi)$. $\square$

Let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, $u \in \mathcal{U}$ be a user, $r \in traces(L)$ be an L-run, $\phi \in RC_{bool}$ is a sentence, and $1 \leq i \leq |r|$. Furthermore, let s be the i-th state of r. The judgment $r, i \vdash_u \phi$ is *data-secure for M, u, and s*, denoted by $secure^{data}_{P,u}(r, i \vdash_u \phi)$, iff for all $s', s'' \in [\![pState(s)]\!]^{data}_{u,M}$, $[\phi]^{s'.db} = [\phi]^{s''.db}$, where $\cong^{data}_{u,M}$ is the data-indistinguishability relation defined in Appendix D and $[\![s]\!]^{data}_{u,M} := \{s' \in \Pi_M | s \cong^{data}_{u,M} s'\}$.

We first recall our definitions and notations for assignments. Let **dom** be the universe and **var** be an infinite countably set of variable identifiers. An *assignment* ν is a partial function from **var** to **dom** that maps variables to values in the universe. Given a formula ϕ and an assignment ν, we say that ν *is well-formed for* ϕ iff ν is defined for all variables in $free(\phi)$. Given an assignment ν and a sequence of variables $\overline{x}$ such that ν is defined for each $x \in \overline{x}$, we denote by $\nu(\overline{x})$ the tuple obtained by replacing each occurrence of $x \in \overline{x}$ with $\nu(x)$. Given an assignment ν, a variable $v \in$ **var**, and a value $u \in$ **dom**, we denote by $\nu \oplus [v \mapsto u]$ the assignment ν' obtained as follows: $\nu'(v') = \nu(v')$ for any $v' \neq v$, and $\nu'(v) = u$. Finally, given a formula ϕ with free variables $free(\phi)$ and an assignment ν, we denote by $\phi \circ \nu$ the formula ϕ' obtained by replacing, for each free variable $x \in free(\phi)$ such that $\nu(x)$ is defined, all the free occurrences of x with $\nu(x)$.

Lemma $F.2$ shows that $secure^{data}_{P,u}$ is a sound, under approximation of $secure_{P,u}$. However, as shown in [24], deciding whether $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds for a given judgment is still undecidable for the relational calculus.

Lemma F.2. *Let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, $u \in \mathcal{U}$ be a user, $r \in traces(L)$ be an L-run, $\phi \in RC_{bool}$ is a sentence, and $1 \leq i \leq |r|$. Given a judgment $r, i \vdash_u \phi$, if $secure^{data}_{P,u}(r, i \vdash_u \phi)$, then $secure_{P,u}(r, i \vdash_u \phi)$.*

Proof. We prove the claim by contradiction. Let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, $u \in \mathcal{U}$ be a user, $r \in traces(L)$ be an L-run, $\phi \in RC_{bool}$ is a sentence, and $1 \leq i \leq |r|$. Furthermore, let $s = \langle db, U, sec, T, V, c\rangle$ be the i-th state of r. Assume, for contradiction's sake, that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds and $secure_{P,u}(r, i \vdash_u \phi)$ does not hold. We denote, for brevity's sake, the fact that $secure_{P,u}(r, i \vdash_u \phi)$ does not hold as $\neg secure_{P,u}(r, i \vdash_u \phi)$. From $\neg secure_{P,u}(r, i \vdash_u \phi)$, it follows that there is a run $r' \in traces(L)$, whose last state is $s' = \langle db', U', sec', T', V', c'\rangle$, such that $r^{i} \cong_{P,u} r'$ and $[\phi]^{db} \neq [\phi]^{db'}$. From the (P, u)-indistinguishability definition, it follows that $pState(last(r^{i}))$ and $pState(last(r'))$ are data indistinguishable according to M and u, i.e., $pState(last(r^{i})) \cong^{data}_{u,M} pState(last(r'))$. From $secure^{data}_{P,u}(r, i \vdash_u \phi)$, it also follows that for all $s', s'' \in [\![pState(s)]\!]^{data}_{u,M}$, $[\phi]^{s'.db} = [\phi]^{s''.db}$. From this and the fact that $pState(last(r^{i})) \cong^{data}_{u,M} pState(last(r'))$, it follows that $[\phi]^{db} = [\phi]^{db'}$, which contradicts $[\phi]^{db} \neq [\phi]^{db'}$. This completes the proof. $\square$

We now show that the rewritings $\phi^{\top}_{s,u}$ and $\phi^{\bot}_{s,u}$ provide the desired properties. First, Lemma $F.3$ proves that the two rewriting satisfy the following invariants: "if $\phi^{\top}_{s,u}$ holds in s, then also ϕ holds in s" and "if $\phi^{\bot}_{s,u}$ does not hold in s, then also ϕ does not hold in s". Afterwards, Lemma $F.4$ shows that both $\phi^{\top}_{s,u}$ and $\phi^{\bot}_{s,u}$ are secure. Then, Lemma $F.5$ shows that $\phi^{\top}_{s,u}$ and $\phi^{\bot}_{s,u}$ are equivalent to $\phi^{\top}_{s',u}$ and $\phi^{\bot}_{s',u}$ for any two data indistinguishable M-state s and s'. Finally, Lemma $F.6$ shows that both $\phi^{\top}_{s,u}$ and $\phi^{\bot}_{s,u}$ are domain-independent.

LEMMA F.3. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be a partial M-state, $u \in U$ be a user, and ϕ be a D-formula. For all assignments ν over **dom** that are well-formed for ϕ, the following conditions hold:*

- *if $[\phi^{\top}_{s,u} \circ \nu]^{db} = \top$, then $[\phi \circ \nu]^{db} = \top$, and*
- *if $[\phi^{\bot}_{s,u} \circ \nu]^{db} = \bot$, then $[\phi \circ \nu]^{db} = \bot$.*

PROOF. Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be a partial M-state, $u \in U$ be a user, and ϕ be a D-formula. Furthermore, let ν be an assignment that is well-formed for ϕ. We prove our claim by induction on the structure of the formula ϕ.

Base Case There are four cases:

1. $\phi := x = y$. In this case, $\phi^{\top}_{s,u} = \phi^{\bot}_{s,u} = \phi$. From this, it follows that $[(x = v)^{\top}_{s,u} \circ \nu]^{db} = [(x = v) \circ \nu]^{db}$ and $[(x = v)^{\bot}_{s,u} \circ \nu]^{db} = [(x = v) \circ \nu]^{db}$. Therefore, our claim follows trivially.
2. $\phi := \top$. The proof of this case is similar to that of $\phi := x = y$.
3. $\phi := \bot$. The proof of this case is similar to that of $\phi := x = y$.
4. $\phi := R(\overline{x})$. Let $\overline{t}$ be the tuple $\nu(\overline{x})$. Note that since ν is well-formed for ϕ, $\overline{t}$ is well-defined.
   Assume that $[\phi^{\top}_{s,u} \circ \nu]^{db} = \top$. From this and $\phi^{\top}_{s,u} := \bigvee_{S \in R^{\top}_{s,u}} S(\overline{x})$, it follows that there is an $S \in R^{\top}_{s,u}$ such that $\overline{t} \in db(S)$. Since $S \in R^{\top}_{s,u}$, it follows that $S \subseteq_M R$. From $S \subseteq_M R$, $\overline{t} \in db(S)$, and Lemma $F.1$, it follows that $\overline{t} \in db(R)$. From this and the relational calculus semantics, it follows that $[\phi \circ \nu]^{db} = \top$.
   Assume that $[\phi^{\bot}_{s,u} \circ \nu]^{db} = \bot$. From this and $\phi^{\bot}_{s,u} := \bigwedge_{S \in R^{\bot}_{s,u}} S(\overline{x})$, it follows that there is an $S \in R^{\bot}_{s,u}$ such that $\overline{t} \notin db(S)$. Since $S \in R^{\bot}_{s,u}$, it follows that $R \subseteq_M S$. From $R \subseteq_M S$, $\overline{t} \notin db(S)$, and Lemma $F.1$, it follows that $\overline{t} \notin db(R)$. From this and the relational calculus semantics, it follows that $[\phi \circ \nu]^{db} = \bot$.

This completes the proof of the base case.

Induction Step Assume that our claim holds for all formulae whose length is less than ϕ's length. We now show that our claim holds also for ϕ. There are a number of cases depending on ϕ's structure.

1. $\phi := \psi \wedge \gamma$. Assume that $[\phi^{\top}_{s,u} \circ \nu]^{db} = \top$. From this and $\phi^{\top}_{s,u} := \psi^{\top}_{s,u} \wedge \gamma^{\top}_{s,u}$, it follows that $[\psi^{\top}_{s,u} \circ \nu]^{db} = \top$ and $[\gamma^{\top}_{s,u} \circ \nu]^{db} = \top$. Since ν is well-formed for ϕ, it is also well-formed for ψ and γ because $free(\psi) \subseteq free(\phi)$ and $free(\gamma) \subseteq free(\phi)$. From $[\psi^{\top}_{s,u} \circ \nu]^{db} = \top$ and the induction hypothesis, it follows that $[\psi \circ \nu]^{db} = \top$. From $[\gamma^{\top}_{s,u} \circ \nu]^{db} = \top$ and the induction hypothesis, it follows that $[\gamma \circ \nu]^{db} = \top$. From $[\psi \circ \nu]^{db} = \top$, $[\gamma \circ \nu]^{db} = \top$, $\phi := \psi \wedge \gamma$, and the relational calculus semantics, it follows that $[\phi \circ \nu]^{db} = \top$.
   Assume that $[\phi^{\bot}_{s,u} \circ \nu]^{db} = \bot$. From this and $\phi^{\bot}_{s,u} := \psi^{\bot}_{s,u} \wedge \gamma^{\bot}_{s,u}$, there are two cases:
   (a) $[\psi^{\bot}_{s,u} \circ \nu]^{db} = \bot$. From $[\psi^{\bot}_{s,u} \circ \nu]^{db} = \bot$ and the induction hypothesis, it follows that $[\psi \circ \nu]^{db} = \bot$. From this, $\phi := \psi \wedge \gamma$, and the relational calculus semantics, it follows that $[\phi \circ \nu]^{db} = \bot$
   (b) $[\gamma^{\bot}_{s,u} \circ \nu]^{db} = \bot$. From $[\gamma^{\bot}_{s,u} \circ \nu]^{db} = \bot$ and the induction hypothesis, it follows that $[\gamma \circ \nu]^{db} = \bot$. From this, $\phi := \psi \wedge \gamma$, and the relational calculus semantics, it follows that $[\phi \circ \nu]^{db} = \bot$
2. $\phi := \psi \vee \gamma$. The proof of this case is similar to that of $\phi := \psi \wedge \gamma$.
3. $\phi := \neg\psi$. Assume that $[\phi^{\top}_{s,u} \circ \nu]^{db} = \top$. From this and $\phi^{\top}_{s,u} := \neg\psi^{\bot}_{s,u}$, it follows that $[\psi^{\bot}_{s,u} \circ \nu]^{db} = \bot$. From this and the induction hypothesis, it follows that $[\psi \circ \nu]^{db} = \bot$. From this, $\phi := \neg\psi$, and the relational calculus semantics, it follows that $[\phi \circ \nu]^{db} = \top$.
   Assume that $[\phi^{\bot}_{s,u} \circ \nu]^{db} = \bot$. From this and $\phi^{\bot}_{s,u} := \neg\psi^{\top}_{s,u}$, it follows that $[\psi^{\top}_{s,u} \circ \nu]^{db} = \top$. From this and the induction hypothesis, it follows that $[\psi \circ \nu]^{db} = \top$. From this, $\phi := \neg\psi$, and the relational calculus semantics, it follows that $[\phi \circ \nu]^{db} = \bot$.
4. $\phi := \exists x.\, \psi$. Assume that $[\phi^{\top}_{s,u} \circ \nu]^{db} = \top$. From this and $\phi^{\top}_{s,u} := \exists x.\, \psi^{\top}_{s,u}$, it follows that there is a $v \in \mathbf{dom}$ such that $[\psi^{\top}_{s,u} \circ \nu[x \mapsto v]]^{db} = \top$. Note that since v is well-formed for ϕ, $\nu[x \mapsto v]$ is well-formed for ψ because $\phi := \exists x.\, \psi$. From this, $[\psi^{\top}_{s,u} \circ \nu[x \mapsto v]]^{db} = \top$, and the induction hypothesis, it follows that $[\psi \circ \nu[x \mapsto v]]^{db} = \top$. From this, $\phi := \exists x.\, \psi$, and the relational calculus semantics, it follows that $[\phi \circ \nu]^{db} = \top$.
   Assume that $[\phi^{\bot}_{s,u} \circ \nu]^{db} = \bot$. From this and $\phi^{\bot}_{s,u} := \exists x.\, \psi^{\bot}_{s,u}$, it follows that for all $v \in \mathbf{dom}$, $[\psi^{\bot}_{s,u} \circ \nu[x \mapsto v]]^{db} = \bot$. Note that since v is well-formed for ϕ, $\nu[x \mapsto v]$ is well-formed for ψ because $\phi := \exists x.\, \psi$. From this, $[\psi^{\bot}_{s,u} \circ \nu[x \mapsto v]]^{db} = \bot$, and the induction hypothesis, it follows that for all $v \in \mathbf{dom}$, $[\psi \circ \nu[x \mapsto v]]^{db} = \bot$. From this, $\phi := \exists x.\, \psi$, and the relational calculus semantics, it follows that $[\phi \circ \nu]^{db} = \bot$.
5. $\phi := \forall x.\, \psi$. The proof of this case is similar to that of $\phi := \exists x.\, \psi$.

This completes the proof of the induction step.

This completes the proof of our claim. □

In Lemma $F.4$, we prove that our rewritings are secure.

LEMMA F.4. *Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP, $r \in traces(L)$ be a run, ϕ be a RC-formula, and $1 \le i \le r$. Furthermore, let s be the i-th state of r. For all assignments ν over **dom** that are well-formed for ϕ, $secure^{data}_{P,u}(r, i \vdash_u \phi^{\top}_{s,u} \circ \nu)$, $secure^{data}_{P,u}(r, i \vdash_u \phi^{\bot}_{s,u} \circ \nu)$, and $secure^{data}_{P,u}(r, i \vdash_u \phi^{rw}_{s,u} \circ \nu)$ hold.*

PROOF. The security of $r, i \vdash_u \phi^{rw}_{s,u}$ follows trivially from that of $r, i \vdash_u \phi^{\top}_{s,u}$ and $r, i \vdash_u \phi^{\bot}_{s,u}$. Therefore, in the following we prove just that $secure^{data}_{P,u}(r, i \vdash_u \phi^{\top}_{s,u} \circ \nu)$ and $secure^{data}_{P,u}(r, i \vdash_u \phi^{\bot}_{s,u} \circ \nu)$ hold. Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be a partial M-state, $u \in U$ be a user, and ϕ be a D-formula. Furthermore, let ν be an assignment that is well-formed for ϕ. We prove our claim by induction on the structure of the formula ϕ.

Base Case There are four cases:

1. $\phi := x = y$. The claim holds trivially. Indeed, $\phi^{\top}_{s,u} \circ \nu$ and $\phi^{\bot}_{s,u} \circ \nu$ are always equivalent either to $\top$ or to $\bot$.

Since for all $s', s'' \in [\![pState(last(r^i))]\!]^{data}_{u,M}$, $[\top]^{s'.db} = [\top]^{s''.db}$ and $[\bot]^{s'.db} = [\bot]^{s''.db}$, it follows that both $secure^{data}_{P,u}(r, i \vdash_u \phi^{\top}_{s,u} \circ \nu)$ and $secure^{data}_{P,u}(r, i \vdash_u \phi^{\bot}_{s,u} \circ \nu)$ hold.

2. $\phi := \top$. The proof of this case is similar to that of $\phi := x = y$.
3. $\phi := \bot$. The proof of this case is similar to that of $\phi := x = y$.
4. $\phi := R(\overline{x})$. Assume, for contradiction's sake, that $secure^{data}_{P,u}(r, i \vdash_u \phi^{\top}_{s,u} \circ \nu)$ does not hold. From this and $secure^{data}_{P,u}$'s definition, it follows that there are two M-partial states $s' = \langle db', U, sec, T, V\rangle$ and $s'' = \langle db'', U, sec, T, V\rangle$ in $[\![pState(last(r^i))]\!]^{data}_{u,M}$ such that $[\phi^{\top}_{s,u} \circ \nu]^{db'} \neq [\phi^{\top}_{s,u} \circ \nu]^{db''}$. Note that this rule out the cases in which $R^{v}_{s,u} = \emptyset$ for any $v \in \{\top, \bot\}$. We assume without loss of generality that $[\phi^{\top}_{s,u} \circ \nu]^{db'} = \top$ and $[\phi^{\top}_{s,u} \circ \nu]^{db''} = \bot$. From this and $\phi^{\top}_{s,u} := \bigvee_{S \in R^{\top}_{s,u}} S(\overline{x})$, it follows that there is an predicate symbol S in the extended vocabulary such that $\nu(\overline{x}) \in db'(S)$ and $\nu(\overline{x}) \notin db''(S)$. There are two cases:
   - S is a table in D or a view in V. Since $S \in R^{\top}_{s,u}$, it follows that $\langle \oplus, \texttt{SELECT}, S\rangle \in permissions(last(r^i), u)$. Note that $permissions(s', u) = permissions(s'', u) = permissions(last(r^i), u)$ because all the states are in the same equivalence class. From $s' \cong^{data}_{u,M} s''$, $\langle \oplus, \texttt{SELECT}, S\rangle \in permissions(s', u)$, and the definition of data indistinguishability, it follows that $db'(S) = db''(S)$. From this, it follows that $\nu(\overline{x}) \in db'(S)$ iff $\nu(\overline{x}) \in db''(S)$, which contradicts $\nu(\overline{x}) \in db'(S)$ and $\nu(\overline{x}) \notin db''(S)$.
   - S is a projection of O, which is either a table in D or a view in V. From $S \in R^{\top}_{s,u}$ and $R^{\top}_{s,u}$'s definition, it follows that $\langle \oplus, \texttt{SELECT}, O\rangle \in permissions(last(r^i), u)$. From $s' \cong^{data}_{u,M} s''$, $\langle \oplus, \texttt{SELECT}, O\rangle \in permissions(s', u)$, and the definition of data indistinguishability, it follows that $db'(O) = db''(O)$. From this and the definition of S, it also follows that $db'(S) = db''(S)$[4]. From this, it follows that $\nu(\overline{x}) \in db'(S)$ iff $\nu(\overline{x}) \in db''(S)$, which contradicts $\nu(\overline{x}) \in db'(S)$ and $\nu(\overline{x}) \notin db''(S)$.

   The proof of $secure^{data}_{P,u}(r, i \vdash_u \phi^{\bot}_{s,u} \circ \nu)$ is analogous.

This completes the proof of the base case.

Induction Step Assume that our claim holds for all formulae whose length is less than ϕ's length. We now show that our claim holds also for ϕ. There are a number of cases depending on ϕ's structure.

1. $\phi := \psi \wedge \gamma$. Assume, for contradiction's sake, that $secure^{data}_{P,u}(r, i \vdash_u \phi^{\top}_{s,u} \circ \nu)$ does not hold. From this and $secure^{data}_{P,u}$'s definition, it follows that there are two M-partial states $s' = \langle db', U, sec, T, V\rangle$ and $s'' = \langle db'', U, sec, T, V\rangle$ in $[\![pState(last(r^i))]\!]^{data}_{u,M}$ such that $[\phi^{\top}_{s,u} \circ \nu]^{db'} \neq [\phi^{\top}_{s,u} \circ \nu]^{db''}$. We assume, without loss of generality, that $[\phi^{\top}_{s,u} \circ \nu]^{db'} = \top$ and $[\phi^{\top}_{s,u} \circ \nu]^{db''} = \bot$. From this and $\phi^{\top}_{s,u} = \psi^{\top}_{s,u} \wedge \gamma^{\top}_{s,u}$, it follows that either $[\psi^{\top}_{s,u} \circ \nu]^{db'} = \top$ and $[\psi^{\top}_{s,u} \circ \nu]^{db''} = \bot$ or $[\gamma^{\top}_{s,u} \circ \nu]^{db'} = \top$ and $[\gamma^{\top}_{s,u} \circ \nu]^{db''} = \bot$. We assume, without loss of generality, that $[\psi^{\top}_{s,u} \circ \nu]^{db'} = \top$ and $[\psi^{\top}_{s,u} \circ \nu]^{db''} = \bot$. From this, it follows that $secure^{data}_{P,u}(r, i \vdash_u \psi^{\top}_{s,u} \circ \nu)$ does not hold. From the induction hypothesis, it follows that $secure^{data}_{P,u}(r, i \vdash_u \psi^{\top}_{s,u} \circ \nu)$ holds leading to a contradiction.
   The proof of $secure^{data}_{P,u}(r, i \vdash_u \phi^{\bot}_{s,u} \circ \nu)$ is analogous.
2. $\phi := \psi \vee \gamma$. The proof of this case is similar to that of $\phi := \psi \wedge \gamma$.
3. $\phi := \neg\psi$. Assume, for contradiction's sake, that $secure^{data}_{P,u}(r, i \vdash_u \phi^{\top}_{s,u} \circ \nu)$ does not hold. From this and $secure^{data}_{P,u}$'s definition, it follows that there are two M-partial states $s' = \langle db', U, sec, T, V\rangle$ and $s'' = \langle db'', U, sec, T, V\rangle$ in $[\![pState(last(r^i))]\!]^{data}_{u,M}$ such that $[\phi^{\top}_{s,u} \circ \nu]^{db'} \neq [\phi^{\top}_{s,u} \circ \nu]^{db''}$. We assume, without loss of generality, that $[\phi^{\top}_{s,u} \circ \nu]^{db'} = \top$ and $[\phi^{\top}_{s,u} \circ \nu]^{db''} = \bot$. From this and $\phi^{\top}_{s,u} = \neg\psi^{\bot}_{s,u}$, it follows that $[\psi^{\bot}_{s,u} \circ \nu]^{db'} = \bot$ and $[\psi^{\bot}_{s,u} \circ \nu]^{db''} = \top$. From this, it follows that $secure^{data}_{P,u}(r, i \vdash_u \psi^{\bot}_{s,u} \circ \nu)$ does not hold. From the induction hypothesis and $\phi := \neg\psi$, it follows that $secure^{data}_{P,u}(r, i \vdash_u \psi^{\bot}_{s,u} \circ \nu)$ holds leading to a contradiction.
   The proof of $secure^{data}_{P,u}(r, i \vdash_u \phi^{\bot}_{s,u} \circ \nu)$ is analogous.
4. $\phi := \exists x.\, \psi$. Assume, for contradiction's sake, that $secure^{data}_{P,u}(r, i \vdash_u \phi^{\top}_{s,u} \circ \nu)$ does not hold. From this and $secure^{data}_{P,u}$'s definition, it follows that there are two M-partial states $s' = \langle db', U, sec, T, V\rangle$ and $s'' = \langle db'', U, sec, T, V\rangle$ in $[\![pState(last(r^i))]\!]^{data}_{u,M}$ such that $[\phi^{\top}_{s,u} \circ \nu]^{db'} \neq [\phi^{\top}_{s,u} \circ \nu]^{db''}$. We assume, without loss of generality, that $[\phi^{\top}_{s,u} \circ \nu]^{db'} = \top$ and $[\phi^{\top}_{s,u} \circ \nu]^{db''} = \bot$. From this and $\phi^{\top}_{s,u} = \exists x.\, \psi^{\top}_{s,u}$, it follows that there is a $v' \in \mathbf{dom}$ such that $[\psi^{\top}_{s,u} \circ \nu[x \mapsto v']]^{db'} = \top$ and there is no $v'' \in \mathbf{dom}$ such that $[\psi^{\top}_{s,u} \circ \nu[x \mapsto v'']]^{db''} = \top$. Therefore, $[\psi^{\top}_{s,u} \circ \nu[x \mapsto v']]^{db'} = \top$ and $[\psi^{\top}_{s,u} \circ \nu[x \mapsto v']]^{db''} = \bot$. Note that $\nu[x \mapsto v']$ is well-formed for $\psi^{\top}_{s,u}$. From this, it follows that $secure^{data}_{P,u}(r, i \vdash_u \psi^{\top}_{s,u} \circ \nu[x \mapsto v'])$ does not hold. However, from the fact that $\nu[x \mapsto v']$ is well-formed for $\psi^{\top}_{s,u}$ and the induction hypothesis, it follows that $secure^{data}_{P,u}(r, i \vdash_u \psi^{\top}_{s,u} \circ \nu[x \mapsto v'])$ holds leading to a contradiction.
   The proof of $secure^{data}_{P,u}(r, i \vdash_u \phi^{\bot}_{s,u} \circ \nu)$ is analogous.
5. $\phi := \forall x.\, \psi$. The proof of this case is similar to that of $\phi := \exists x.\, \psi$.

This completes the proof of the induction step.

This completes the proof of our claim. □

PROPOSITION F.1. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ and $s' = \langle db', U', sec', T', V'\rangle$ be two partial M-states, $u \in U$ be a user, $v \in \{\top, \bot\}$, and ϕ be a D-formula. If $s \cong^{data}_{u,M} s'$, then $bound(\phi, s, u, x, v) = bound(\phi, s', u, x, v)$.*

PROOF. Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ and $s' = \langle db', U', sec', T', V'\rangle$ be two partial M-states, $u \in U$ be a user, $v \in \{\top, \bot\}$, and ϕ be a D-formula. We prove our claim by induction on the structure of the formula ϕ.

Base Case There are four cases:

1. $\phi := y = z$. The result of $bound(\phi, s, u, x, v)$ and $bound(\phi, s', u, x, v)$ does not depend on s. Therefore, $bound(\phi, s, u, x, v) = bound(\phi, s', u, x, v)$.
2. $\phi := \top$. $bound(\phi, s, u, x, v) = bound(\phi, s', u, x, v) = \bot$.
3. $\phi := \bot$. $bound(\phi, s, u, x, v) = bound(\phi, s', u, x, v) = \bot$.

[4]With a slight abuse of notation, we consider S as a view.

4. $\phi := R(\overline{x})$. The result of $bound(\phi, s, u, x, v)$ and $bound(\phi, s', u, x, v)$ depend only on the sets $R^v_{s,u}$ and $R^v_{s',u}$, which in turn depend on the content of the sets R^v_s, $R^v_{s'}$, $AUTH^*_{s,u}$, and $AUTH^*_{s',u}$. Assume that $s \cong^{data}_{u,M} s'$. From this, it follows that $R^v_s = R^v_{s'}$ (because D is the same and $V = V'$) and $AUTH^*_{s,u} = AUTH^*_{s',u}$ (because $sec = sec'$). From this, it follows that $bound(\phi, s, u, x, v) = bound(\phi, s', u, x, v)$.

This completes the proof of the base case.

Induction Step Assume that our claim holds for all formulae whose length is less than ϕ. We now show that our claim holds also for ϕ. There are a number of cases depending on ϕ's structure.

1. $\phi := \psi \wedge \gamma$. Assume that $s \cong^{data}_{u,M} s'$. From this and the induction hypothesis, it follows that $bound(\psi, s, u, x, v) = bound(\psi, s', u, x, v)$ and $bound(\gamma, s, u, x, v) = bound(\gamma, s', u, x, v)$. From this and $bound(\phi, s, u, x, v) := bound(\psi, s, u, x, v) \vee bound(\gamma, s, u, x, v)$, it follows that $bound(\phi, s, u, x, v) = bound(\phi, s', u, x, v)$.
2. $\phi := \psi \vee \gamma$. The proof of this case is similar to that of $\phi := \psi \wedge \gamma$.
3. $\phi := \neg\psi$. Assume that $s \cong^{data}_{u,M} s'$. From this and the induction hypothesis, it follows that $bound(\psi, s, u, x, v) = bound(\psi, s', u, x, v)$. From this, $bound(\neg\psi, s, u, x, v) = bound(\psi, s, u, x, \neg v)$, and $bound(\neg\psi, s', u, x, v) = bound(\psi, s', u, x, \neg v)$, it follows that $bound(\phi, s, u, x, v) = bound(\phi, s', u, x, v)$.
4. $\phi := \exists y.\, \psi$. Assume that $s \cong^{data}_{u,M} s'$. There are two cases:
   (a) $x = y$. In this case, the proof is trivial as $bound(\phi, s, u, x, v) = bound(\phi, s', u, x, v) = \bot$.
   (b) $x \neq y$. In this case, $bound(\phi, s, u, x, v) = bound(\psi, s, u, x, v) \wedge bound(\psi, s, u, y, v)$ and $bound(\phi, s', u, x, v) = bound(\psi, s', u, x, v) \wedge bound(\psi, s', u, y, v)$. From $s \cong^{data}_{u,M} s'$ and the induction hypothesis, it follows that $bound(\psi, s, u, x, v) = bound(\psi, s', u, x, v)$ and $bound(\psi, s, u, y, v) = bound(\psi, s', u, y, v)$. From this, $bound(\phi, s, u, x, v) = bound(\psi, s, u, x, v) \wedge bound(\psi, s, u, y, v)$, and $bound(\phi, s', u, x, v) = bound(\psi, s', u, x, v) \wedge bound(\psi, s', u, y, v)$, it follows that $bound(\phi, s, u, x, v) = bound(\phi, s', u, x, v)$.
5. $\phi := \forall x.\, \psi$. The proof of this case is similar to that of $\phi := \exists x.\, \psi$.

This completes the proof of the induction step.

This completes the proof of our claim. □

LEMMA F.5. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ and $s' = \langle db', U', sec', T', V'\rangle$ be two partial M-states, $u \in U$ be a user, and ϕ be a D-formula. If $s \cong^{data}_{u,M} s'$, then $\phi^{\top}_{s,u} = \phi^{\top}_{s',u}$, $\phi^{\bot}_{s,u} = \phi^{\bot}_{s',u}$, and $\phi^{rw}_{s,u} = \phi^{rw}_{s',u}$.*

PROOF. Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ and $s' = \langle db', U', sec', T', V'\rangle$ be two partial M-states, $u \in U$ be a user, and ϕ be a D-formula. We prove our claim by induction on the structure of the formula ϕ.

Base Case There are four cases:

1. $\phi := x = y$. The claim holds trivially. Indeed, $\phi^{\top}_{s,u} = \phi^{\bot}_{s,u} = \phi$.
2. $\phi := \top$. The proof of this case is similar to that of $\phi := x = y$.
3. $\phi := \bot$. The proof of this case is similar to that of $\phi := x = y$.
4. $\phi := R(\overline{x})$. The formulae $\phi^{\top}_{s,u}$ and $\phi^{\top}_{s',u}$ depend only on the sets $R^{\top}_{s,u}$ and $R^{\top}_{s',u}$, which in turn depends on $R^{\top}_s$, $R^{\top}_{s'}$, $AUTH^*_{s,u}$, and $AUTH^*_{s',u}$. If $s \cong^{data}_{u,M} s'$, then $R^{\top}_s = R^{\top}_{s'}$ (because D is the same and $V = V'$) and $AUTH^*_{s,u} = AUTH^*_{s',u}$ (because $sec = sec'$). Therefore, $\phi^{\top}_{s,u} = \phi^{\top}_{s',u}$. The proof for $\phi^{\bot}_{s,u}$ is analogous.

This completes the proof of the base case.

Induction Step Assume that our claim holds for all formulae whose length is less than ϕ. We now show that our claim holds also for ϕ. There are a number of cases depending on ϕ's structure.

1. $\phi := \psi \wedge \gamma$. Assume that $s \cong^{data}_{u,M} s'$. From this and the induction hypothesis, it follows that $\psi^{\top}_{s,u} = \psi^{\top}_{s',u}$ and $\gamma^{\top}_{s,u} = \gamma^{\top}_{s',u}$. From this, $\phi := \psi \wedge \gamma$, $\phi^{\top}_{s,u} := \psi^{\top}_{s,u} \wedge \gamma^{\top}_{s,u}$, and $\phi^{\top}_{s',u} := \psi^{\top}_{s',u} \wedge \gamma^{\top}_{s',u}$, it follows that $\phi^{\top}_{s,u} = \phi^{\top}_{s',u}$. The proof of $\phi^{\bot}_{s,u} = \phi^{\bot}_{s',u}$ is analogous.
2. $\phi := \psi \vee \gamma$. The proof of this case is similar to that of $\phi := \psi \wedge \gamma$.
3. $\phi := \neg\psi$. Assume that $s \cong^{data}_{u,M} s'$. From this and the induction hypothesis, it follows that $\psi^{\top}_{s,u} = \psi^{\top}_{s',u}$ and $\psi^{\bot}_{s,u} = \psi^{\bot}_{s',u}$. From this, $\phi := \neg\psi$, $\phi^{\top}_{s,u} := \neg\psi^{\bot}_{s,u}$, and $\phi^{\top}_{s',u} := \neg\psi^{\bot}_{s',u}$, it follows that $\phi^{\top}_{s,u} = \phi^{\top}_{s',u}$. The proof of $\phi^{\bot}_{s,u} = \phi^{\bot}_{s',u}$ is analogous.
4. $\phi := \exists x.\, \psi$. Assume that $s \cong^{data}_{u,M} s'$. From this and the induction hypothesis, it follows that $\psi^{\top}_{s,u} = \psi^{\top}_{s',u}$. We remark that $bound(\psi, s, u, x, \top) = bound(\psi, s', u, x, \top)$, as proved in Proposition $F.1$. There are two cases:
   (a) $bound(\psi, s, u, x, \top) = \top$. From this, $bound(\psi, s, u, x, \top) = bound(\psi, s', u, x, \top)$, $\psi^{\top}_{s,u} = \psi^{\top}_{s',u}$, $\phi := \exists x.\, \psi$, $\phi^{\top}_{s,u} := \exists x.\, \psi^{\top}_{s,u}$, and $\phi^{\top}_{s',u} := \exists x.\, \psi^{\top}_{s',u}$, it follows that $\phi^{\top}_{s,u} = \phi^{\top}_{s',u}$.
   (b) $bound(\psi, s, u, x, \top) = \bot$. From this, $bound(\psi, s, u, x, \top) = bound(\psi, s', u, x, \top)$, and $\phi^{\top}_{s,u}$ definition, it follows that $\phi^{\top}_{s',u} = \phi^{\top}_{s',u} = \bot$.

   The proof of $\phi^{\bot}_{s,u} = \phi^{\bot}_{s',u}$ is analogous.
5. $\phi := \forall x.\, \psi$. The proof of this case is similar to that of $\phi := \exists x.\, \psi$.

This completes the proof of the induction step.

The equivalence $\phi^{rw}_{s,u} = \phi^{rw}_{s',u}$ follows trivially from $\phi^{rw}_{s,u}$'s definition and the fact that $\phi^{\top}_{s,u} = \phi^{\top}_{s',u}$ and $\phi^{\bot}_{s,u} = \phi^{\bot}_{s',u}$. This completes the proof of our claim. □

Before proving the domain independence of $\phi^{\top}_{s,u}$ and $\phi^{\bot}_{s,u}$, we introduce some notation. The relation *gen*, introduced in [45], is the smallest relation defined by the rules in Figure 40. Note that we extended *gen* by adding the rules *Equiv*, *Const 1*, and *Const 2*. A relational calculus formula ϕ is *allowed* iff it satisfies the following conditions:

- for all $x \in free(\phi)$, $gen(x, \phi)$ holds,
- for every sub-formula $\exists x.\psi$ in ϕ, $gen(x, \psi)$ holds, and
- for every sub-formula $\forall x.\psi$ in ϕ, $gen(x, \neg\psi)$ holds.

As shown in [45], every allowed formula is domain independent. Note that the addition of the *Equiv*, *Const 1*, and *Const 2* rules does not modify this result.

PROPOSITION F.2. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be an M-partial state, $u \in U$ be a user, and $v \in \{\top, \bot\}$. For any formulae ϕ and ψ, the following equivalences hold:*

- $(\neg\phi)^v_{s,u} \equiv \neg\phi^{\neg v}_{s,u}$,

$$\frac{x \in \overline{x}}{gen(x, R(\overline{x}))}\ \text{Pred} \qquad \frac{gen(x, push(\neg\phi))}{gen(x, \neg\phi)}\ \text{Neg}$$

$$\frac{x \neq y \quad gen(x,\phi)}{gen(x, \exists y.\phi)}\ \text{Exists} \qquad \frac{x \neq y \quad gen(x,\phi)}{gen(x, \forall y.\phi)}\ \text{For all}$$

$$\frac{gen(x,\phi) \quad gen(x,\psi)}{gen(x, \phi \vee \psi)}\ \text{Or} \qquad \frac{gen(x,\psi) \quad \phi \equiv \psi}{gen(x, \phi)}\ \text{Equiv}$$

$$\frac{v \in \mathbf{dom}}{gen(x, x = v)}\ \text{Const 1} \qquad \frac{v \in \mathbf{dom}}{gen(x, v = x)}\ \text{Const 2}$$

$$\frac{gen(x,\phi)}{gen(x, \phi \wedge \psi)}\ \text{And 1} \qquad \frac{gen(x,\psi)}{gen(x, \phi \wedge \psi)}\ \text{And 2}$$

$$push(\phi) = \begin{cases} \neg\psi \vee \neg\gamma & \text{if } \phi := \neg(\psi \wedge \gamma) \\ \neg\psi \wedge \neg\gamma & \text{if } \phi := \neg(\psi \vee \gamma) \\ \forall x.\, \neg\psi & \text{if } \phi := \neg\exists x.\psi \\ \exists x.\, \neg\psi & \text{if } \phi := \neg\forall x.\psi \\ \psi & \text{if } \phi := \neg\neg\psi \\ x \neq y & \text{if } \phi := \neg(x = y) \\ x = y & \text{if } \phi := \neg(x \neq y) \end{cases}$$

Figure 40: *gen* rules

- $(\phi)^v_{s,u} \wedge (\psi)^v_{s,u} \equiv (\phi \wedge \psi)^v_{s,u}$,
- $(\phi)^v_{s,u} \vee (\psi)^v_{s,u} \equiv (\phi \vee \psi)^v_{s,u}$,
- $(\exists x.\, \phi)^v_{s,u} \equiv (\neg\forall x.\, \neg\phi)^v_{s,u}$, *and*
- $(\forall x.\, \phi)^v_{s,u} \equiv (\neg\exists x.\, \neg\phi)^v_{s,u}$.

PROOF. Let $M = \langle D, \Gamma \rangle$ be a system configuration, $s = \langle db, U, sec, T, V \rangle$ be an M-partial state, $u \in U$ be a user, $v \in \{\top, \bot\}$, and ϕ, ψ be two formulae.

- $(\neg\phi)^v_{s,u} \equiv \neg\phi^{\neg v}_{s,u}$. This case follows trivially from the definition of the rewriting.
- $(\phi)^v_{s,u} \wedge (\psi)^v_{s,u} \equiv (\phi \wedge \psi)^v_{s,u}$. This case follows trivially from the definition of the rewriting.
- $(\phi)^v_{s,u} \vee (\psi)^v_{s,u} \equiv (\phi \vee \psi)^v_{s,u}$ This case follows trivially from the definition of the rewriting.
- $(\exists x.\, \phi)^v_{s,u} \equiv (\neg\forall x.\, \neg\phi)^v_{s,u}$. There are two cases:
  1. $bound(\phi, s, u, x, v) = \top$. From this, it follows that $(\exists x.\, \phi)^v_{s,u} = \exists x.\, \phi^v_{s,u}$. From $(\neg\phi)^v_{s,u} \equiv \neg\phi^{\neg v}_{s,u}$ and $(\neg\forall x.\, \neg\phi)^v_{s,u}$, it follows that $(\neg\forall x.\, \neg\phi)^v_{s,u} \equiv \neg(\forall x.\, \neg\phi)^{\neg v}_{s,u}$. From the definition of *bound*, it follows that $bound(\neg\phi, s, u, x, \neg v) = bound(\phi, s, u, x, \neg\neg v)$. From this and $v = \neg\neg v$, it follows that $bound(\neg\phi, s, u, x, \neg v) = bound(\phi, s, u, x, v)$. From this and $bound(\phi, s, u, x, v) = \bot$, it follows that $bound(\neg\phi, s, u, x, \neg v) = \top$. From this, it follows that $(\forall x.\, \neg\phi)^{\neg v}_{s,u} = \forall x.\, (\neg\phi)^{\neg v}_{s,u}$. From this and $(\neg\phi)^v_{s,u} \equiv \neg\phi^{\neg v}_{s,u}$, it follows that $(\forall x.\, \neg\phi)^{\neg v}_{s,u} \equiv \forall x.\, \neg\phi^v_{s,u}$. From this and $(\neg\forall x.\, \neg\phi)^v_{s,u} \equiv \neg(\forall x.\, \neg\phi)^{\neg v}_{s,u}$, it follows that $(\neg\forall x.\, \neg\phi)^v_{s,u} \equiv \neg\forall x.\, \neg\phi^v_{s,u}$. From this and standard RC equivalences, it follows that $(\neg\forall x.\, \neg\phi)^v_{s,u} \equiv \exists x.\, \phi^v_{s,u}$.
  2. $bound(\phi, s, u, x, v) = \bot$. From this, it follows that $(\exists x.\, \phi)^v_{s,u} = \neg v$. From $(\neg\phi)^v_{s,u} \equiv \neg\phi^{\neg v}_{s,u}$ and $(\neg\forall x.\, \neg\phi)^v_{s,u}$, it follows that $(\neg\forall x.\, \neg\phi)^v_{s,u} \equiv \neg(\forall x.\, \neg\phi)^{\neg v}_{s,u}$. From the definition of *bound*, it follows that $bound(\neg\phi, s, u, x, \neg v) = bound(\phi, s, u, x, \neg\neg v)$. From this and $v = \neg\neg v$, it follows that $bound(\neg\phi, s, u, x, \neg v) = bound(\phi, s, u, x, v)$. From this and $bound(\phi, s, u, x, v) = \bot$, it follows that $bound(\neg\phi, s, u, x, \neg v) = \bot$. From this, it follows that $(\forall x.\, \neg\phi)^{\neg v}_{s,u} = v$. From this and $(\neg\forall x.\, \neg\phi)^v_{s,u} \equiv \neg(\forall x.\, \neg\phi)^{\neg v}_{s,u}$, it follows that $(\neg\forall x.\, \neg\phi)^v_{s,u} \equiv \neg v$. From this and $(\exists x.\, \phi)^v_{s,u} = \neg v$, it follows that $(\exists x.\, \phi)^v_{s,u} \equiv (\neg\forall x.\, \neg\phi)^v_{s,u}$.
- $(\forall x.\, \phi)^v_{s,u} \equiv (\neg\exists x.\, \neg\phi)^v_{s,u}$. The proof of this case is similar to that of $(\exists x.\, \phi)^v_{s,u} \equiv (\neg\forall x.\, \neg\phi)^v_{s,u}$.

This completes the proof. □

PROPOSITION F.3. *Let $M = \langle D, \Gamma \rangle$ be a system configuration, $s = \langle db, U, sec, T, V \rangle$ be an M-partial state, $u \in U$ be a user, $v \in \{\top, \bot\}$, and ϕ be a formula. Furthermore, let $x \in free(\phi) \cap free(\phi^v_{s,u})$. If $gen(x, \phi)$ holds, then $gen(x, \phi^v_{s,u})$ holds.*

PROOF. Let $M = \langle D, \Gamma \rangle$ be a system configuration, $s = \langle db, U, sec, T, V \rangle$ be an M-partial state, $u \in U$ be a user, $v \in \{\top, \bot\}$, and ϕ be a formula. Furthermore, let $x \in free(\phi) \cap free(\phi^v_{s,u})$. We prove our claim by structural induction on the length of ϕ. In the following, the length of ϕ is the number of predicates, quantifiers, negations, conjunctions, and disjunctions in ϕ.

Base Case There are four cases:

1. $\phi := x = y$. In this case, the claim holds trivially.
2. $\phi := \top$. In this case, the claim holds trivially.
3. $\phi := \bot$. In this case, the claim holds trivially.
4. $\phi := R(\overline{x})$. Assume $gen(x, \phi)$ holds. From this, it follows that x is one of the free variables in $\overline{x}$. Furthermore, from $x \in free(\phi^v_{s,u})$, it follows that $R^v_{s,u} \neq \emptyset$. There are two cases:
   - (a) $\phi^v_{s,u}$ is a conjunction of predicates $S(\overline{x})$ such that $gen(x, S(\overline{x}))$ holds. From the rule *And 1*, it follows that $gen(x, \phi^v_{s,u})$ holds.
   - (b) $\phi^v_{s,u}$ is a disjunction of predicates $S(\overline{x})$ such that $gen(x, S(\overline{x}))$ holds. From the rule *Or*, it follows that $gen(x, \phi^v_{s,u})$ holds.

This completes the proof of the base case.

Induction Step Assume that our claim holds for all formulae whose length is less than ϕ's length. We now show that our claim holds also for ϕ. There are a number of cases depending on ϕ's structure.

1. $\phi := \psi \wedge \gamma$. Assume that $gen(x, \phi)$ holds. From this and the rules *And 1* and *And 2*, it follows that either $gen(x, \psi)$ or $gen(x, \gamma)$ hold. Assume, without loss of generality, that $gen(x, \psi)$ holds. From this and the induction hypothesis, it follows that $gen(x, \psi^v_{s,u})$ holds. From this, $\phi^v_{s,u} := \psi^v_{s,u} \wedge \gamma^v_{s,u}$, and the rule *And 1*, it follows that $gen(x, \phi^v_{s,u})$ holds.
2. $\phi := \psi \vee \gamma$. Assume that $gen(x, \phi)$ holds. From this and the rule *Or*, it follows that both $gen(x, \psi)$ and $gen(x, \gamma)$ hold. From this and the induction hypothesis, it follows that both $gen(x, \psi^v_{s,u})$ and $gen(x, \gamma^v_{s,u})$ hold. From this, $\phi^v_{s,u} := \psi^v_{s,u} \vee \gamma^v_{s,u}$, and the rule *Or*, it follows that $gen(x, \phi^v_{s,u})$ holds.
3. $\phi := \neg\psi$. Assume that $gen(x, \phi)$ holds. From this and the rule *Not*, it follows that $gen(x, push(\neg\psi))$. There are a number of cases depending on ψ. In the following, we exploit standard relational calculus equivalences, see, for instance, [3], and the equivalences we proved in Proposition *F.2*.
   - (a) $\psi := (\alpha \wedge \beta)$. In this case, $push(\neg\psi)$ is $(\neg\alpha \vee \neg\beta)$. From this and $gen(x, push(\neg\psi))$, it follows $gen(x, (\neg\alpha \vee \neg\beta))$. From this and the *Or* rule, it follows that $gen(x, \neg\alpha)$ and $gen(x, \neg\beta)$ hold. From this and the induction hypothesis, it follows that $gen(x, (\neg\alpha)^v_{s,u})$ and $gen(x, (\neg\beta)^v_{s,u})$. From this and the *Or* rule, it follows that $gen(x, (\neg\alpha)^v_{s,u} \vee (\neg\beta)^v_{s,u})$.

From this, $(\neg\alpha)^v_{s,u} \vee (\neg\beta)^v_{s,u} \equiv (\neg\alpha \vee \neg\beta)^v_{s,u}$, and the *Equiv* rule, it follows that $gen(x, (\neg\alpha\vee\neg\beta)^v_{s,u})$. From this, $(\neg\alpha \vee \neg\beta)^v_{s,u} \equiv (\neg(\alpha \wedge \beta))^v_{s,u}$, and the *Equiv* rule, it follows that $gen(x, (\neg(\alpha \wedge \beta))^v_{s,u})$. From this, $(\neg(\alpha \wedge \beta))^v_{s,u} \equiv \neg(\alpha \wedge \beta)^{\neg v}_{s,u}$, and the *Equiv* rule, it follows that $gen(x, \neg(\alpha\wedge\beta)^{\neg v}_{s,u})$. From this and $\psi := \alpha \wedge \beta$, it follows that $gen(x, \neg\psi^{\neg v}_{s,u})$. From this and $\phi^v_{s,u} := \neg\psi^{\neg v}_{s,u}$, it follows that $gen(x, \phi^v_{s,u})$ holds.

(b) $\psi := (\alpha \vee \beta)$. The proof is similar to the $\psi := \neg(\alpha \wedge \beta)$ case.

(c) $\psi := \exists y.\, \alpha$. In this case, $push(\neg\psi)$ is $\forall y.\, \neg\alpha$. From this and $gen(x, push(\neg\psi))$, it follows $gen(x, \forall y.\, \neg\alpha)$. From this and the induction hypothesis, it follows that $gen(x, (\forall y.\, \neg\alpha)^v_{s,u})$. From this, $\neg\neg(\forall y.\, \neg\alpha)^v_{s,u} \equiv (\forall y.\, \neg\alpha)^v_{s,u}$, and the *Equiv* rule, it follows that $gen(x, \neg\neg(\forall y.\, \neg\alpha)^v_{s,u})$. From this, $\neg\neg(\forall y.\, \neg\alpha)^v_{s,u} \equiv \neg(\neg\forall y.\, \neg\alpha)^{\neg v}_{s,u}$, and the *Equiv* rule, it follows that $gen(x, \neg(\neg\forall y.\, \neg\alpha)^{\neg v}_{s,u})$. From this, $\neg(\neg\forall y.\, \neg\alpha)^{\neg v}_{s,u} \equiv \neg(\exists y.\, \neg\neg\alpha)^{\neg v}_{s,u}$, and the *Equiv* rule, it follows that $gen(x, \neg(\exists y.\, \neg\neg\alpha)^{\neg v}_{s,u})$. From this, $\neg(\exists y.\, \neg\neg\alpha)^{\neg v}_{s,u} \equiv \neg(\exists y.\, \alpha)^{\neg v}_{s,u}$, and the *Equiv* rule, it follows that $gen(x, \neg(\exists y.\, \alpha)^{\neg v}_{s,u})$. From this and $\psi := \exists y.\, \alpha$, it follows that $gen(x, \neg\psi^{\neg v}_{s,u})$. From this and $\phi^v_{s,u} := \neg\psi^{\neg v}_{s,u}$, it follows that $gen(x, \phi^v_{s,u})$ holds.

(d) $\psi := \forall y.\, \alpha$. The proof is similar to the $\psi := \neg\exists y.\, \alpha$ case.

(e) $\psi := \neg\alpha$. In this case, $push(\neg\psi)$ is α. From this and $gen(x, push(\neg\psi))$, it follows $gen(x, \alpha)$. From this and the induction hypothesis, it follows that $gen(x, \alpha^v_{s,u})$. From this, $\neg\neg\alpha^v_{s,u} \equiv \alpha^v_{s,u}$, and the *Equiv* rule, it follows that $gen(x, \neg\neg\alpha^v_{s,u})$. From this, $\neg\neg\alpha^v_{s,u} \equiv \neg(\neg\alpha)^{\neg v}_{s,u}$, and the *Equiv* rule, it follows that $gen(x, \neg(\neg\alpha)^{\neg v}_{s,u})$. From this and $\psi := \neg\alpha$, it follows that $gen(x, \neg\psi^{\neg v}_{s,u})$. From this and $\phi^v_{s,u} := \neg\psi^{\neg v}_{s,u}$, it follows that $gen(x, \phi^v_{s,u})$ holds.

(f) $\psi := x = y$. The proof for this case is trivial.

(g) $\psi := x \neq y$. The proof for this case is trivial.

4. $\phi := \exists x.\, \psi$. Assume that $gen(x, \phi)$ holds. From this and the rule *Exists*, it follows that $gen(x, \psi)$ holds. From this and the induction hypothesis, it follows that $gen(x, \psi^v_{s,u})$ holds. From this, $\phi^v_{s,u} := \exists x.\, \psi^v_{s,u}$, and the rule *Exists*, it follows that $gen(x, \phi^v_{s,u})$ holds.
5. $\phi := \forall x.\, \psi$. The proof of this case is similar to that of $\phi := \exists x.\, \psi$.

This completes the proof of the induction step.

This completes the proof of our claim. □

Proposition F.4. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be an M-partial state, $u \in U$ be a user, $v \in \{\top, \bot\}$, and ϕ be a formula. For every sub-formula $\exists x.\, \psi$ of ϕ, if $gen(x, \psi)$ holds and $x \in free(\psi) \cap free(\psi^v_{s,u})$, then $gen(x, \psi^v_{s,u})$ holds.*

Proof. Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be an M-partial state, $u \in U$ be a user, $v \in \{\top, \bot\}$, and ϕ be a formula. We prove our claim by structural induction on the length of ϕ. In the following, the size of ϕ is the number of predicates, quantifiers, negations, conjunctions, and disjunctions in ϕ.

Base Case The claim is vacuously satisfied for the base cases as there is no sub-formula of the form $\exists x.\, \psi$.

Induction Step Assume that our claim holds for all formulae whose length is less than ϕ. We now show that our claim holds also for ϕ. There are a number of cases depending on ϕ's structure.

1. $\phi := \psi \wedge \gamma$. Let α be a sub-formula of ϕ of the form $\exists x.\, \beta$ such that $gen(x, \beta)$ holds and $x \in free(\beta) \cap free(\beta^v_{s,u})$. The formula α is either a sub-formula of ψ or a sub-formula of γ. From this and the induction hypothesis, it follows that $gen(x, \beta^s_{v,u})$ holds.
2. $\phi := \psi \vee \gamma$. The proof of this case is similar to that of $\phi := \psi \wedge \gamma$.
3. $\phi := \neg\psi$. Let α be a sub-formula of ϕ of the form $\exists x.\, \beta$ such that $gen(x, \beta)$ holds and $x \in free(\beta) \cap free(\beta^v_{s,u})$. Since $\phi := \neg\psi$, the formula α is also a sub-formula of ψ. From this and the induction hypothesis, it follows that $gen(x, \beta^s_{v,u})$ holds.
4. $\phi := \exists x.\, \psi$. Let α be a sub-formula of ϕ of the form $\exists x.\, \beta$ such that $gen(x, \beta)$ holds and $x \in free(\beta)\cap free(\beta^v_{s,u})$. There are two cases:
   (a) α is a sub-formula of ψ. From this and the induction hypothesis, it follows that $gen(x, \beta^s_{v,u})$ holds.
   (b) $\alpha = \phi$. From $gen(x, \beta)$, $x \in free(\beta)\cap free(\beta^v_{s,u})$, and Proposition F.3, it follows that $gen(x, \beta^v_{s,u})$ holds.
5. $\phi := \forall x.\, \psi$. The proof of this case is similar to that of $\phi := \exists x.\, \psi$.

This completes the proof of the induction step.

This completes the proof of our claim. □

Proposition F.5. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be an M-partial state, $u \in U$ be a user, $v \in \{\top, \bot\}$, and ϕ be a formula. For every sub-formula $\forall x.\, \psi$ of ϕ, if $gen(x, \psi)$ holds and $x \in free(\psi) \cap free(\psi^v_{s,u})$, then $gen(x, (\neg\psi)^v_{s,u})$ holds.*

Proof. The proof is similar to that of Proposition F.4. □

Proposition F.6. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be an M-partial state, $u \in U$ be a user, $v \in \{\top, \bot\}$, and ϕ be a formula. Let $Q \in \{\exists, \forall\}$ be a quantifier and $subs_Q(\phi)$ be the set of sub-formulae of ϕ of the form $Q\, x.\, \psi$. There is a surjective function f from $subs_Q(\phi)$ to $subs_Q(\phi^v_{s,u})$ such that for any $Q\, x.\, \psi$ in $subs_Q(\phi)$, if $f(Q\, x.\, \psi)$ is defined, then $f(Q\, x.\, \psi)^v_{s,u} = Q\, x.\, \psi^v_{s,u}$.*

Proof. The claim follows trivially from the definition of $\phi^v_{s,u}$. □

Lemma F.6. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be an M-partial state, $u \in U$ be a user, and ϕ be a formula. If ϕ is allowed and all views in V are allowed, then $\phi^\top_{s,u}$, $\phi^\bot_{s,u}$, and $\phi^{rw}_{s,u}$ are domain independent.*

Proof. From Proposition F.3, Proposition F.4, Proposition F.5, and Proposition F.6, it follows that if ϕ is allowed, then both $\phi^\top_{s,u}$ and $\phi^\bot_{s,u}$ are allowed. Since every allowed formula is domain independent [45], it follows that both $\phi^\top_{s,u}$ and $\phi^\bot_{s,u}$ are domain independent. Finally, the domain independence of $\phi^{rw}_{s,u}$ follows easily from its definition and the domain independence of $\phi^\top_{s,u}$ and $\phi^\bot_{s,u}$. □

We now prove the main result of this section, namely that the *secure* function is, indeed, a sound, under-approximation of the notion of judgment's security.

Lemma F.7. *Let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, $u \in \mathcal{U}$ be a user, $r \in traces(L)$ be an L-run, $\phi \in RC_{bool}$ is a sentence, and $1 \le i \le |r|$. Furthermore, let s be the i-th state in r. The following statements hold:*

1. *Given a judgment $r, i \vdash_u \phi$, if $secure(u, \phi, s) = \top$, then $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds.*
2. *Given a judgment $r, i \vdash_u \phi$, if $secure(u, \phi, s) = \top$, then $secure_{P,u}(r, i \vdash_u \phi)$ holds.*

PROOF. Note that the second statement follows trivially from Lemma $F.2$ and the first statement. Therefore, in the following we prove just that given a judgment $r, i \vdash_u \phi$, if $secure(u, \phi, s) = \top$, then $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds.

Let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, $u \in \mathcal{U}$ be a user, $r \in traces(L)$ be an L-run, $\phi \in RC_{bool}$ is a sentence, and $1 \leq i \leq |r|$. Furthermore, let $s = \langle db, U, sec, T, V, c\rangle$ be the i-th state in r. Assume that $secure(u, \phi, s) = \top$. From this and *secure*'s definition, $[\phi^{rw}_{s,u}]^{db} = \bot$. In the following, with a slight abuse of notation we ignore the *inline* and *ext* functions in $\phi^{rw}_{s,u}$'s definition. This is without loss of generality since *inline* and *ext* do not modify ϕ's result. From this and $\phi^{rw}_{s,u}$'s definition, it follows that either $[\phi^{\top}_{s,u}]^{db} = \top$ or $[\phi^{\bot}_{s,u}]^{db} = \bot$. Note that from Lemma $F.4$, it follows that $secure^{data}_{P,u}(r, i \vdash_u \phi^{\top}_{s,u})$ and $secure^{data}_{P,u}(r, i \vdash_u \phi^{\bot}_{s,u})$. Furthermore, let Δ be the equivalence class $[\![pState(s)]\!]^{data}_{u,M}$. There are two cases:

1. $[\phi^{\top}_{s,u}]^{db} = \top$. From $secure^{data}_{P,u}(r, i \vdash_u \phi^{\top}_{s,u})$, it follows that for all $s', s'' \in \Delta$, $[\phi^{\top}_{s,u}]^{s'.db} = [\phi^{\top}_{s,u}]^{s''.db}$. From this, $s \in \Delta$, and $[\phi^{\top}_{s,u}]^{db} = \top$, it follows that $[\phi^{\top}_{s,u}]^{s'.db} = \top$ for all $s' \in \Delta$. From Lemma $F.5$, it follows that for all $s', s'' \in \Delta$, $\phi^{\top}_{s,u} = \phi^{\top}_{s',u} = \phi^{\top}_{s'',u}$. From this and the fact that for all $s' \in \Delta$, $[\phi^{\top}_{s,u}]^{s'.db} = \top$, it follows that for all $s' \in \Delta$, $[\phi^{\top}_{s',u}]^{s'.db} = \top$. From this and Lemma $F.3$, it follows that for all $s' \in \Delta$, $[\phi]^{s'.db} = \top$. From this, it follows that for all $s', s'' \in \Delta$, $[\phi]^{s'.db} = [\phi]^{s''.db}$. From this, r's definition, and $secure^{data}_{P,u}$, it follows that $secure^{data}_{P,u}(r, i \vdash_u \phi)$.
2. $[\phi^{\bot}_{s,u}]^{db} = \bot$. From $secure^{data}_{P,u}(r, i \vdash_u \phi^{\bot}_{s,u})$, it follows that for all $s', s'' \in \Delta$, $[\phi^{\bot}_{s,u}]^{s'.db} = [\phi^{\bot}_{s,u}]^{s''.db}$. From this, $s \in \Delta$, and $[\phi^{\bot}_{s,u}]^{db} = \bot$, it follows that $[\phi^{\bot}_{s,u}]^{s'.db} = \bot$ for all $s' \in \Delta$. From Lemma $F.5$, it follows that for all $s', s'' \in \Delta$, $\phi^{\bot}_{s,u} = \phi^{\bot}_{s',u} = \phi^{\bot}_{s'',u}$. From this and the fact that for all $s' \in \Delta$, $[\phi^{\bot}_{s,u}]^{s'.db} = \bot$, it follows that for all $s' \in \Delta$, $[\phi^{\bot}_{s',u}]^{s'.db} = \bot$. From this and Lemma $F.3$, it follows that for all $s' \in \Delta$, $[\phi]^{s'.db} = \bot$. From this, it follows that for all $s', s'' \in \Delta$, $[\phi]^{s'.db} = [\phi]^{s''.db}$. From this, r's definition, and $secure^{data}_{P,u}$, it follows that $secure^{data}_{P,u}(r, i \vdash_u \phi)$.

This completes the proof of our claim. $\square$

Lemma $F.8$ proves that the *secure* function produces the same result for any two indistinguishable states.

LEMMA F.8. *Let M be a system configuration, $u \in U$ be a user, $s, s' \in \Omega_M$ be two M-states such that $pState(s) \cong^{data}_{u,M} pState(s')$, and ϕ be a sentence. Then, $secure(u, \phi, s) = \top$ iff $secure(u, \phi, s') = \top$.*

PROOF. Let M be a system configuration, $u \in U$ be a user, $s = \langle db, U, sec, T, V, c\rangle$ and $s' = \langle db', U', sec', T', V', c'\rangle$ be two M-states such that $pState(s) \cong^{data}_{u,M} pState(s')$, and ϕ be a sentence. We now prove that $secure(u, \phi, s) = secure(u, \phi, s')$. Assume, for contradiction's sake, that $secure(u, \phi, s) \neq secure(u, \phi, s')$. From this, it follows that $[\phi^{rw}_{s,u}]^{db} \neq [\phi^{rw}_{s',u}]^{db'}$. From $pState(s) \cong^{data}_{u,M} pState(s')$ and Lemma $F.5$, it follows that $\phi^{rw}_{s,u} = \phi^{rw}_{s',u}$. From this and $[\phi^{rw}_{s,u}]^{db} \neq [\phi^{rw}_{s',u}]^{db'}$, it follows that $[\phi^{rw}_{s,u}]^{db} \neq [\phi^{rw}_{s,u}]^{db'}$. This contradicts $secure^{data}_{P,u}(r, i \vdash_u \phi^{rw}_{s,u})$, which has been proved in Lemma $F.4$. This completes the proof of our claim. $\square$

F.2 Data Confidentiality Proofs

In this section, we first prove some simple results about f^u_{conf}. Afterwards, we prove our main result, namely that f^u_{conf} provides data confidentiality with respect to the user u.

LEMMA F.9. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $u \in \mathcal{U}$ be a user, $s, s' \in \Omega_M$ be two M-states such that $pState(s) \cong^{data}_{u,M} pState(s')$, $invoker(s) = invoker(s')$, and $tr(s) = tr(s')$, and a be an action in $\mathcal{A}_{D,\mathcal{U}}$. Then, $f^u_{conf}(s, a) = f^u_{conf}(s', a)$.*

PROOF. Let $M = \langle D, \Gamma\rangle$ be a system configuration, $u \in \mathcal{U}$ be a user, $s = \langle db, U, sec, T, V, c\rangle$ and $s' = \langle db', U', sec', T', V', c'\rangle$ be two M-states such that $pState(s) \cong^{data}_{u,M} pState(s')$, $invoker(s) = invoker(s')$, and $tr(s) = tr(s')$, and a be an action in $\mathcal{A}_{D,u}$. There are a number of cases depending on the action a.

1. $a = \langle u', \texttt{SELECT}, \phi\rangle$. Assume, for contradiction's sake, that $f^u_{conf,}(s, a) \neq f^u_{conf}(s', a)$. This happens iff $secure(u, \phi, s) \neq secure(u, \phi, s')$. This contradicts Lemma $F.8$ because $pState(s) \cong^{data}_{u,M} pState(s')$.
2. $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$. We claim that $noLeak(s, a, u) = noLeak(s', a, u)$. Assume, for contradiction's sake, that $f^u_{conf,}(s, a) \neq f^u_{conf}(s', a)$. This happens iff there is a formula ϕ, which has been derived using the *getInfo*, *getInfoV*, or *getInfoD* functions, such that $secure(u, \phi, s) \neq secure(u, \phi, s')$. This contradicts Lemma $F.8$ because $pState(s) \cong^{data}_{u,M} pState(s')$.
   We prove our claim that $noLeak(s, a, u) = noLeak(s', a, u)$ for any two states s and s' such that $pState(s) \cong^{data}_{u,M} pState(s')$. Assume, for contradiction's sake, that this is not the case. Without loss of generality we assume that $noLeak(s, a, u) = \top$ and $noLeak(s', a, u) = \bot$. From $noLeak(s, a, u) = \top$, it follows that for all views V such that $\langle\oplus, \texttt{SELECT}, V\rangle \in permissions(s, u)$ and $R \in tDet(V, s, M)$, for all $o \in tDet(V, s, M)$, $\langle\oplus, \texttt{SELECT}, o\rangle$ is in $permissions(s, u)$. From $pState(s) \cong^{data}_{u,M} pState(s')$, it follows that $sec = sec'$. From this, $permissions(s, u) = permissions(s', u)$. From $noLeak(s', a, u) = \bot$, there are two views V' and o such that $\langle\oplus, \texttt{SELECT}, V'\rangle \in permissions(s', u)$, $\langle\oplus, \texttt{SELECT}, o\rangle \notin permissions(s', u)$, $R \in tDet(V', s', M)$, and $o \in tDet(V', s', M)$. Note that $tDet(V', s', M) = tDet(V', s, M)$ because query determinacy does not consider the database state. From this and $permissions(s, u) = permissions(s', u)$, it follows that there is a view V' such that $\langle\oplus, \texttt{SELECT}, V'\rangle \in permissions(s, u)$ and $R \in tDet(V', s, M)$, such that there is a table $o \in tDet(V', s, M)$ for which $\langle\oplus, \texttt{SELECT}, o\rangle \notin permissions(s, u)$. This contradicts $noLeak(s, a, u) = \top$.
3. $a = \langle u', \texttt{DELETE}, R, \overline{t}\rangle$. The proof of this case is similar to the $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$ case.
4. $a = \langle op, u'', p, u'\rangle$, where $op \in \{\oplus, \oplus^*\}$. Assume, for contradiction's sake, that $f^u_{conf,}(s, a) \neq f^u_{conf}(s', a)$. Note that this happens iff $p = \langle \texttt{SELECT}, o\rangle$ for some o. Without loss of generality, we further assume that $f^u_{conf,}(s, a) = \top$ and $f^u_{conf}(s', a) = \bot$. From $f^u_{conf,}(s, a) = \top$, it follows that $\langle\oplus, \texttt{SELECT}, o\rangle \in permissions(s, u)$. From

$pState(s) \cong^{data}_{u,M} pState(s')$, it follows $permissions(s,u) = permissions(s',u)$. From this and $\langle \oplus, \texttt{SELECT}, o\rangle \in permissions(s,u)$, it follows that $\langle \oplus, \texttt{SELECT}, o\rangle$ is in $permissions(s',u)$. From $f^u_{conf,}(s',a) = \bot$, it follows that $\langle \oplus, \texttt{SELECT}, o\rangle \notin permissions(s,u)$. This contradicts $\langle \oplus, \texttt{SELECT}, o\rangle \in permissions(s',u)$.

5. For any other action a, the proof is trivial.

This completes the proof of our claim. □

Lemma F.10. *Let P be an extended configuration, L be the P-LTS, $r \in traces(L)$ be a run, u be a user, γ be a sentence, and Φ be a set of sentences such that $\Phi \models_{fin} \gamma$. If, for all $\phi \in \Phi$, $secure_{P,u}(r, i \vdash_u \phi)$ holds and $[\phi]^{last(r).db} = \top$, then $secure_{P,u}(r, i \vdash_u \gamma)$ holds and $[\gamma]^{last(r^i).db} = \top$.*

Proof. Let P be an extended configuration, L be the P-LTS, $r \in traces(L)$ be a run, u be a user, γ be a sentence, and Φ be a set of sentences such that $\Phi \models_{fin} \gamma$ such that for all $\phi \in \Phi$, $secure_{P,u}(r, i \vdash_u \phi)$ holds and $[\phi]^{last(r^i).db} = \top$. We now show that $secure_{P,u}(r, i \vdash_u \gamma)$ holds and $[\gamma]^{last(r^i).db} = \top$. From $\Phi \models_{fin} \gamma$, the fact that for all $\phi \in \Phi$, $[\phi]^{last(r^i).db} = \top$, and $\models_{fin}$'s definition, it follows that $[\gamma]^{last(r^i).db} = \top$. Assume, for contradiction's sake, that $secure_{P,u}(r, i \vdash_u \gamma)$ does not hold. From this and $[\gamma]^{last(r^i).db} = \top$, it follows that there is a run $r' \in traces(L)$ such that $r^i \cong_{P,u} r'$ such that $[\gamma]^{last(r').db} = \bot$. We claim that for all $\phi \in \Phi$, $[\phi]^{last(r').db} = \top$. From this and $\Phi \models_{fin} \gamma$, it follows that $[\gamma]^{last(r').db} = \top$, which contradicts $[\gamma]^{last(r').db} = \bot$.

We now prove our claim that for all $\phi \in \Phi$, $[\phi]^{last(r').db} = \top$ for any trace r' such that $r^i \cong_{P,u} r'$. From $secure_{P,u}(r, i \vdash_u \phi)$, it follows that $[\phi]^{last(r^i).db} = [\phi]^{last(r').db}$. From this and $[\phi]^{last(r^i).db} = \top$, it follows that $[\phi]^{last(r').db} = \top$. □

Before proving our main result, namely that f^u_{conf} provides data confidentiality for the user u, we introduce the concept of an action that preserves the equivalence class induced by the indistinguishability relation $\cong_{P,u}$.

Definition F.3. Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP, L be the P-LTS, $r \in traces(L)$ be a run, a be an action in $\mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D$, and u be a user in $\mathcal{U}$. We denote by $extend(r,a)$, where r is a run and a is an action, the run $r' \in traces(L)$, where $s \in \Omega_M$ and $r' = r \cdot a \cdot s$, obtained by executing the action a at the end of the run r'. If there is no such run, then $extend(r,a)$ is undefined. We say that *a preserves the equivalence class for r, P, and u* iff (1) $extend(r,a)$ is defined, and (2) there is a bijection b between $[\![r]\!]_{P,u}$ and $[\![extend(r,a)]\!]_{P,u}$ such that for all $r' \in [\![r]\!]_{P,u}$, $extend(r',a)$ is defined and $b(r') = extend(r',a)$. □

Lemma F.11. *Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP, L be the P-LTS, u be a user in $\mathcal{U}$, r be a run in $traces(L)$, $a \in \mathcal{A}_{D,u}$ be an `INSERT` or `DELETE` action $\langle u, op, R, \overline{t}\rangle$, ϕ be a sentence, and i be such that $1 \le i \le |r|$, $triggers(last(r^i)) = \epsilon$, and $r^{i+1} = extend(r^i, a)$. If (1) a preserves the equivalence class for r^i, P, and u, and (2) the execution of a does not change any table in $tables(\phi)$ for any run $v \in [\![r^i]\!]_{P,u}$, then $secure_{P,u}(r, i \vdash_u \phi)$ holds iff $secure_{P,u}(r, i+1 \vdash_u \phi)$ holds.*

Proof. Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP, L be the P-LTS, u be a user in $\mathcal{U}$, r be a run in $traces(L)$, $a \in \mathcal{A}_{D,u}$ be an `INSERT` or `DELETE` action $\langle u, op, R, \overline{t}\rangle$, ϕ be a sentence, and i be such that $1 \le i \le |r|$, $triggers(last(r^i)) = \epsilon$, and $r^{i+1} = extend(r^i, a)$. Assume that (1) a preserves the equivalence class for r^i, P, and u, and (2) the execution of a does not change any table in $tables(\phi)$ for any run $v \in [\![r^i]\!]_{P,u}$. Without loss of generality, assume that a is an `INSERT` action. In the following, we denote the *extend* function by e. Furthermore, we also denote the fact that $secure_{P,u}(r,i,u,\phi)$ does not hold as $\neg secure_{P,u}(r,i,u,\phi)$. From Definition F.3 and a preserves the equivalence class for r^i, P, and u, it follows that $e(r',a)$ is defined for any $r' \in [\![r^i]\!]_{P,u}$. Assume, for contradiction's sake, that our claim does not hold. There are two cases:

- $secure_{P,u}(r, i \vdash_u \phi)$ holds and $secure_{P,u}(r, i+1 \vdash_u \phi)$ does not hold. From $secure_{P,u}(r, i \vdash_u \phi)$, it follows that for all $r' \in [\![r^i]\!]_{P,u}$, $[\phi]^{last(r').db} = [\phi]^{last(r^i).db}$. We claim that $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r^i]\!]_{P,u}$. From this and $[\phi]^{last(r').db} = [\phi]^{last(r^i).db}$ for all $r' \in [\![r^i]\!]_{P,u}$, it follows that $[\phi]^{last(r^i).db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r^i]\!]_{P,u}$. From $\neg secure_{P,u}(r, i+1 \vdash_u \phi)$, it follows that there is a run $r' \in [\![r^{i+1}]\!]_{P,u}$ such that $[\phi]^{last(r^{i+1}).db} \neq [\phi]^{last(r').db}$. From this, $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ for any $r' \in [\![r^i]\!]_{P,u}$, and $e(r^i,a) = r^{i+1}$, it follows that $[\phi]^{last(r^i).db} \neq [\phi]^{last(r').db}$. Let b be the bijection showing that a preserves the equivalence class with respect to r^i, P, and u. From $e(r^i,a) = r^{i+1}$ and $r' \in [\![r^{i+1}]\!]_{P,u}$, it follows that $r' \in [\![e(r^i,a)]\!]_{P,u}$. From this, it follows that there is a $r'' = b^{-1}(r')$ such that $r'' \in [\![r^i]\!]_{P,u}$ and $e(r'',a) = r'$. From this and $[\phi]^{last(v).db} = [\phi]^{last(e(v,a)).db}$ for any $v \in [\![r^i]\!]_{P,u}$, it follows that $[\phi]^{last(r'').db} = [\phi]^{last(r').db}$. From this and $[\phi]^{last(r^i).db} \neq [\phi]^{last(r').db}$, it follows that $[\phi]^{last(r^i).db} \neq [\phi]^{last(r'').db}$. This contradicts the fact that for all $r' \in [\![r^i]\!]_{P,u}$, $[\phi]^{last(r').db} = [\phi]^{last(r^i).db}$. Indeed, $r'' \in [\![r^i]\!]_{P,u}$ and $[\phi]^{last(r^i).db} \neq [\phi]^{last(r'').db}$.
  We prove our claim that $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r^i]\!]_{P,u}$. Assume that this is not the case. This implies that the content of one of the relations that determines ϕ is different in $last(r').db$ and $last(e(r',a)).db$. This is impossible. Indeed, if a's execution has been successful, i.e., $secEx(last(e(r',a))) = \bot$ and $Ex(last(e(r',a))) = \emptyset$, then a's execution does not change any table in $tables(\phi)$, and the set of relations that determines ϕ is always a subset of $tables(\phi)$. This leads to a contradiction, and, therefore, $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds. Similarly, if a's execution has not been successful, i.e., $secEx(last(e(r',a))) = \top$ or $Ex(last(e(r',a))) \neq \emptyset$, then $last(r').db$ is the same as $last(e(r',a)).db$, and the claim holds trivially.
- $secure_{P,u}(r, i+1 \vdash_u \phi)$ holds and $secure_{P,u}(r, i \vdash_u \phi)$ does not hold. We have already shown that $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r]\!]_{P,u}$. From $\neg secure_{P,u}(r, i \vdash_u \phi)$, it follows that there is $r' \in [\![r^i]\!]_{P,u}$ such that $[\phi]^{last(r^i).db} \neq [\phi]^{last(r').db}$. Let b the bijection showing that a preserves the equivalence class with respect to r, P, and u. Since $r' \in [\![r^i]\!]_{P,u}$, then let $r'' =$

$b(r') = e(r', a)$. From $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r^i]\!]_{P,u}$, it follows that $[\phi]^{last(r^i).db} \neq [\phi]^{last(e(r',a)).db}$. From this, $e(r^i, a) = r^{i+1}$, and the fact that $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r]\!]_{P,u}$, it follows that $[\phi]^{last(r^{i+1}).db} \neq [\phi]^{last(e(r',a)).db}$. From this and $e(r', a) \in [\![r^{i+1}]\!]_{P,u}$, it follows $\neg secure_{P,u}$ $(r, i{+}1 \vdash_u \phi)$. This contradicts the fact that $secure_{P,u}(r,$ $i + 1 \vdash_u \phi)$ holds.

This completes the proof. $\square$

LEMMA F.12. *Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP, L be the P-LTS, u be a user in $\mathcal{U}$, r be a run in $traces(L)$, $a \in \mathcal{A}_{D,u}$ be a `SELECT` or `CREATE` action, ϕ be a sentence, and i be such that $1 \leq i \leq |r|$, $triggers(last(r^i)) = \epsilon$, and $r^{i+1} = extend(r^i, a)$. If a preserves the equivalence class for r^i, P, and u, then $secure_{P,u}(r, i \vdash_u \phi)$ holds iff $secure_{P,u}(r, i + 1 \vdash_u \phi)$ holds.*

PROOF. Proof similar to that of Lemma $F.11$. $\square$

LEMMA F.13. *Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP, L be the P-LTS, u be a user in $\mathcal{U}$, r be a run in $traces(L)$, $a \in \mathcal{A}_{D,u}$ be a `GRANT` or `REVOKE` action, ϕ be a sentence, and i be such that $1 \leq i \leq |r|$, $triggers(last(r^i)) = \epsilon$, and $r^{i+1} = extend(r^i, a)$. If a preserves the equivalence class for r^i, P, and u, then $secure_{P,u}(r, i \vdash_u \phi)$ holds iff $secure_{P,u}(r, i + 1 \vdash_u \phi)$ holds.*

PROOF. Proof similar to that of Lemma $F.11$. $\square$

LEMMA F.14. *Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP, L be the P-LTS, u be a user in $\mathcal{U}$, r be a run in $traces(L)$, a be a trigger in $\mathcal{TRIGGER}_D$, ϕ be a sentence, and i be such that $1 \leq i \leq |r|$, $invoker(last(r^i)) = u$, and $r^{i+1} = extend(r^i, a)$. If (1) a preserves the equivalence class for r^i, P, and u, (2) if a's action is either an `INSERT` or `DELETE`, then t's execution does not change any table in $tables(\phi)$ for any run $v \in [\![r^i]\!]_{P,u}$, and (3) $secEx(last(extend$ $(r^i, a)) = \bot$ and $Ex(last(extend(r^i, a)) = \emptyset$, then $secure_{P,u}(r,$ $i \vdash_u \phi)$ holds iff $secure_{P,u}(r, i + 1 \vdash_u \phi)$ holds.*

PROOF. Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP, L be the P-LTS, u be a user in $\mathcal{U}$, r be a run in $traces(L)$, a be a trigger in $\mathcal{TRIGGER}_D$, ϕ be a sentence, and i be such that $1 \leq i \leq |r|$, $invoker(last(r^i)) = u$, and $r^{i+1} = extend(r^i, a)$. Assume also (1) that a preserves the equivalence class for r^i, P, and u, and (2) $secEx(last(extend$ $(r^i, a)) = \bot$ and $Ex(last(extend(r^i, a)) = \emptyset$. In the following, we denote the *extend* function by e. Furthermore, we also denote the fact that $secure_{P,u}(r, i \vdash_u \phi)$ does not hold as $\neg secure_{P,u}(r, i \vdash_u \phi)$. From Definition F.3 and the fact that a preserves the equivalence class for r^i, P, and u, it follows that $e(r', a)$ is defined for any $r' \in [\![r^i]\!]_{P,u}$. Assume, for contradiction's sake, that our claim does not hold. There are two cases:

- $secure_{P,u}(r, i \vdash_u \phi)$ holds and $secure_{P,u}(r, i + 1 \vdash_u \phi)$ does not hold. From $secure_{P,u}(r, i \vdash_u \phi)$, it follows that $[\phi]^{last(r^i).db} = [\phi]^{last(r').db}$ for any $r' \in [\![r^i]\!]_{P,C}$. We claim that $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r^i]\!]_{P,u}$. From $\neg secure_{P,u}(r, i + 1 \vdash_u \phi)$, it follows that there is a $r'' \in [\![r^{i+1}]\!]_{P,u}$ such that $[\phi]^{last(r'').db} \neq [\phi]^{last(r^{i+1}).db}$. Let b the bijection showing that a preserves the equivalence class with respect to r^i, P, and u. Since $r^{i+1} = e(r^i, a)$ and $r' \in [\![e(r, a)]\!]_{P,u}$, then there is a run $v \in [\![r^i]\!]_{P,u}$ such that $v = b^{-1}(r'')$. From this, $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r^i]\!]_{P,u}$, and the fact that $[\phi]^{last(r'').db} \neq [\phi]^{last(r^{i+1}).db}$, it follows that $[\phi]^{last(v).db} \neq [\phi]^{last(r^{i+1}).db}$. From this, $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r^i]\!]_{P,u}$, and $r^{i+1} = e(r^i, a)$, it follows $[\phi]^{last(v).db} \neq [\phi]^{last(r^i).db}$. This contradicts the fact that $[\phi]^{last(r^i).db} = [\phi]^{last(r').db}$ for any $r' \in [\![r^i]\!]_{P,C}$.
  We now prove that $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r^i]\!]_{P,u}$. Assume, for contradiction's sake, that there is a run $r' \in [\![r^i]\!]_{P,u}$ such that $[\phi]^{last(r').db} \neq [\phi]^{last(e(r',a)).db}$. There are three cases:
  - the trigger a is not enabled in $e(r', a)$. From this and the LTS semantics, it follows that $last(r').db = last(e(r', a)).db$. From this, it therefore follows that $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$. This contradicts our assumption.
  - the trigger a is enabled in $e(r', a)$ and its action is a `GRANT` or a `REVOKE`. From this and the LTS semantics, it therefore follows that $last(r').db = last(e(r', a)).db$. From this, it thus follows that $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$. This contradicts our assumption.
  - the trigger a is enabled in $e(r', a)$ and its action is a `INSERT` or a `GRANT`. Thus, from $[\phi]^{last(r').db} \neq [\phi]^{last(e(r',a)).db}$, it follows that the content of one of the relations that determines ϕ is different in $last(r').db$ and $last(e(r', a)).db$. This contradicts the fact that the a's execution does not change the tables in $tables(\phi)$ for any run $r' \in [\![r^i]\!]_{P,u}$.
- $secure_{P,u}(r, i + 1 \vdash_u \phi)$ holds and $secure_{P,u}(r, i \vdash_u \phi)$ does not hold. We have already shown that $[\phi]^{last(r').db}$ $= [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r]\!]_{P,u}$. From $\neg secure_{P,u}(r, i \vdash_u \phi)$, it follows that there is $r' \in [\![r^i]\!]_{P,u}$ such that $[\phi]^{last(r^i).db} \neq [\phi]^{last(r').db}$. Let b the bijection showing that a preserves the equivalence class with respect to r, P, and u. Since $r' \in [\![r^i]\!]_{P,u}$, then let $r'' = b(r') = e(r', a)$. From $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r^i]\!]_{P,u}$, it follows that $[\phi]^{last(r^i).db} \neq [\phi]^{last(e(r',a)).db}$. From this, $e(r^i, a) = r^{i+1}$, and the fact that $[\phi]^{last(r').db} = [\phi]^{last(e(r',a)).db}$ holds for any $r' \in [\![r]\!]_{P,u}$, it follows that $[\phi]^{last(r^{i+1}).db} \neq [\phi]^{last(e(r',a)).db}$. From this and $e(r', a) \in [\![r^{i+1}]\!]_{P,u}$, it follows $\neg secure_{P,u}$ $(r, i{+}1 \vdash_u \phi)$. This contradicts $secure_{P,u}(r, i{+}1 \vdash_u \phi)$.

This completes the proof. $\square$

PROPOSITION F.7. *Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP, L be the P-LTS, $a \in \mathcal{A}_{D,u}$ be an `INSERT` or `DELETE` action, and r be a run such that $tr(last(r)) = \epsilon$. For any constraint γ in $Dep(\Gamma, a)$, the following statements hold:*

- $[getInfoS(\gamma, a)]^{last(r).db} = \top$ *iff* $\gamma \notin Ex(last(extend(r, a)))$, *and*
- $[getInfoV(\gamma, a)]^{last(r).db} = \top$ *iff* $\gamma \in Ex(last(extend(r, a)))$.

PROOF. Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is an M-PDP, L be the P-LTS, $a \in \mathcal{A}_{D,u}$ be an `INSERT` or `DELETE` action, and r be a run such that $tr(last(r)) = \epsilon$. Furthermore, let γ be a constraint in $Dep(\Gamma, a)$. We first note that $getInfoS(\gamma, a) = \neg getInfoV(\gamma, a)$. From this, it follows trivially that we can prove just one of the two claims. We thus prove that $[getInfoS(\gamma, a)]^{last(r).db} = \top$ iff $\gamma \notin Ex(last(extend(r, a)))$. There are two cases:

1. $a = \langle u, \texttt{INSERT}, R, \overline{t}\rangle$. There are two cases depending on γ:
   (a) γ is of the form $\forall \overline{x}, \overline{y}, \overline{y}', \overline{z}, \overline{z}'. (R(\overline{x}, \overline{y}, \overline{z}) \wedge R(\overline{x}, \overline{y}', \overline{z}')) \Rightarrow \overline{y} = \overline{y}'$. Let $\overline{t}$ be $(\overline{v}, \overline{w}, \overline{q})$, db be the state $last(r).db$, and db' be the state $db[R \oplus \overline{t}]$.
   ($\Rightarrow$) Assume that $[getInfoS(\gamma, a)]^{last(r).db} = \top$. From this and $getInfoS(\gamma, a)$'s definition, it follows that for all tuples $(\overline{v}, \overline{w}', \overline{q}') \in db(R)$, then $\overline{w}' = \overline{w}$. From a's definition and the LTS semantics, it follows that $db'(R) = db(R) \cup \{(\overline{v}, \overline{w}, \overline{q})\}$. From this and the fact that for all tuples $(\overline{v}, \overline{w}', \overline{q}') \in db(R)$, then $\overline{w}' = \overline{w}$, it follows that for all tuples $(\overline{v}, \overline{w}', \overline{q}') \in db'(R)$, then $\overline{w}' = \overline{w}$. Furthermore, since $db \in \Omega_D^\Gamma$, it follows that for all tuples $(\overline{v}', \overline{w}', \overline{q}'), (\overline{v}'', \overline{w}'', \overline{q}'') \in db'(R)$, if $\overline{v}' = \overline{v}''$ and $\overline{v}' \neq \overline{v}$, then $\overline{w}' = \overline{w}$. Therefore, it follows that for all tuples $(\overline{v}', \overline{w}', \overline{q}')$, $(\overline{v}'', \overline{w}'', \overline{q}'') \in db'(R)$, if $\overline{v}' = \overline{v}''$, then $\overline{w}' = \overline{w}$. Therefore, $[\gamma]^{db'} = \top$. From this and the LTS semantics, it follows that $\gamma \notin Ex(last(extend(r, a)))$.
   ($\Leftarrow$) Assume that $\gamma \notin Ex(last(extend(r, a))))$. From this and the LTS semantics, it follows that $[\gamma]^{db'} = \top$. Therefore, for any two tuples $(\overline{v}', \overline{w}', \overline{q}')$ and $(\overline{v}'', \overline{w}'', \overline{q}'') \in db'(R)$, if $\overline{v}' = \overline{v}''$, then $\overline{w}' = \overline{w}$. Assume, for contradiction's sake, that $[getInfoS(\gamma, a)]^{db} = \bot$. This means that there is a tuple $(\overline{v}, \overline{w}', \overline{q}')$ in $db(R)$ such that $\overline{w}' \neq \overline{w}$. From $db' = db[R(\overline{v}, \overline{w}, \overline{q})]$ and the LTS semantics, it follows that both $(\overline{v}, \overline{w}', \overline{q}')$ and $(\overline{v}, \overline{w}, \overline{q})$ are in $db'(R)$. From this and $\overline{w}' \neq \overline{w}$, it follows that there are two tuples $(\overline{v}, \overline{w}, \overline{q})$ and $(\overline{v}, \overline{w}', \overline{q}')$ in $db(R)$ such that $\overline{w}' \neq \overline{w}$. From this and the relational calculus semantics, it follows that $[\gamma]^{db} = \bot$. This is in contradiction with $[\gamma]^{db'} = \top$.
   (b) γ is of the form $\forall \overline{x}, \overline{z}. R(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. S(\overline{x}, \overline{w})$. Let $\overline{t}$ be $(\overline{v}, \overline{w})$, db be the state $last(r).db$, and db' be the state $db[R \oplus \overline{t}]$.
   ($\Rightarrow$) Assume that $[getInfoS(\gamma, a)]^{db} = \top$. From this and $getInfoS(\gamma, a)$'s definition, it follows that there is a tuple $(\overline{v}, \overline{y})$ in $db(S)$. From a's definition and the LTS semantics, it follows that $db'(S) = db(S)$. From this, it follows that there is a tuple $(\overline{v}, \overline{y})$ in $db'(S)$. Furthermore, since $db \in \Omega_D^\Gamma$, it follows that for all tuples $(\overline{v}', \overline{w}') \in db(R)$, if $\overline{v}' \neq \overline{v}$, there is a tuple $(\overline{v}', \overline{y}') \in db(S)$. From this and $\overline{db}' = db[R \oplus (\overline{v}, \overline{w})]$, it follows that for all tuples $(\overline{v}', \overline{w}') \in db'(R)$, there is a tuple $(\overline{v}', \overline{y}') \in db'(S)$. Therefore, $[\gamma]^{db'} = \top$. From this and the LTS semantics, it follows that $\gamma \notin Ex(last(extend(r, a)))$.
   ($\Leftarrow$) Assume that $\gamma \notin Ex(last(extend(r, a)))$. From this and the LTS semantics, it follows that $[\gamma]^{db'} = \top$. Therefore, for any tuple $(\overline{v}', \overline{w}') \in db'(R)$, there is a tuple $(\overline{v}', \overline{y}') \in db'(S)$. Assume, for contradiction's sake, that $[getInfoS(\gamma, a)]^{db} = \bot$. This means that for any tuple $(\overline{v}', \overline{y}')$ in $db(S)$, $\overline{v}' \neq \overline{v}$. From $db'(S) = db(S)$, it follows that for any tuple $(\overline{v}', \overline{y}')$ in $db'(S)$, $\overline{v}' \neq \overline{v}$. From $db' = db[R \oplus (\overline{v}, \overline{w})]$, it follows that there is a tuple $(\overline{v}, \overline{w})$ in $db'(R)$ such that there is no tuple $(\overline{v}, \overline{y}')$ in $db'(S)$. From this and the relational calculus semantics, it follows that $[\gamma]^{db} = \bot$. This is in contradiction with $[\gamma]^{db'} = \top$.
2. $a = \langle u, \texttt{DELETE}, R, \overline{t}\rangle$. In this case, γ is of the form $\forall \overline{x}, \overline{z}. S(\overline{x}, \overline{z}) \Rightarrow \exists \overline{w}. R(\overline{x}, \overline{w})$. Let $\overline{t}$ be $(\overline{v}, \overline{w})$, db be the state $last(r).db$, and db' be the state $db[R \ominus \overline{t}]$.
   ($\Rightarrow$) Assume that $[getInfoS(\gamma, a)]^{db} = \top$. From this and $getInfoS(\gamma, a)$'s definition, it follows that either there is no tuple $(\overline{v}, \overline{y})$ in $db(S)$ or there is a tuple $(\overline{v}, \overline{w}')$ in $db(R)$ such that $\overline{w}' \neq \overline{w}$. There are two cases:
   (a) there is no tuple $(\overline{v}, \overline{y})$ in $db(S)$. From this, a's definition, and the LTS semantics, it follows that there is no tuple $(\overline{v}, \overline{y})$ in $db'(S)$. From $db \in \Omega_D^\Gamma$, it follows that for all tuples $(\overline{v}', \overline{y}')$ in $db(S)$ such that $\overline{v}' \neq \overline{v}$, there is a tuple $(\overline{v}', \overline{w}')$ in $db(R)$. From this, $db'(R) = db(R) \setminus \{(\overline{v}, \overline{w})\}$, $db'(S) = db(S)$, and there is no tuple $(\overline{v}, \overline{y})$ in $db'(S)$, it follows that for all tuples $(\overline{v}', \overline{y}')$ in $db(S)$, there is a tuple $(\overline{v}', \overline{w}')$ in $db(R)$. Therefore, $[\gamma]^{db'} = \top$. From this and the LTS semantics, it follows that $\gamma \notin Ex(last(extend(r, a)))$.
   (b) there is a tuple $(\overline{v}, \overline{w}')$ in $db(R)$ such that $\overline{w}' \neq \overline{w}$. From this, a's definition, and the LTS semantics, it follows that there is a tuple $(\overline{v}, \overline{w}')$ in $db'(R)$ such that $\overline{w}' \neq \overline{w}$. From $db \in \Omega_D^\Gamma$, it follows that for all tuples $(\overline{v}', \overline{y}')$ in $db(S)$ such that $\overline{v}' \neq \overline{v}$, there is a tuple $(\overline{v}', \overline{w}'')$ in $db(R)$. From this, $db'(R) = db(R) \setminus \{(\overline{v}, \overline{w})\}$, $db'(S) = db(S)$, and there is a tuple $(\overline{v}, \overline{w}')$ in $db'(R)$ such that $\overline{w}' \neq \overline{w}$, it follows that for all tuples $(\overline{v}', \overline{y}')$ in $db(S)$, there is a tuple $(\overline{v}', \overline{w}')$ in $db(R)$. Therefore, $[\gamma]^{db'} = \top$. From this and the LTS semantics, it follows that $\gamma \notin Ex(last(extend(r, a)))$.
   ($\Leftarrow$) Assume that $\gamma \notin Ex(last(extend(r, a)))$. From this and the LTS semantics, it follows that $[\gamma]^{db'} = \top$. Therefore, for any tuple $(\overline{v}', \overline{y}') \in db'(S)$, there is a tuple $(\overline{v}', \overline{w}') \in db'(R)$. Assume, for contradiction's sake, that $[getInfoS(\gamma, a)]^{db} = \bot$. Therefore, there is a tuple $(\overline{v}, \overline{y})$ in $db(S)$ and for all tuples $(\overline{v}, \overline{w}'')$ in $db(R)$, $\overline{w}'' = \overline{w}$. From this, $db'(S) = db(S)$, and $db' = db[R \ominus (\overline{v}, \overline{w})]$, it follows that there is a tuple $(\overline{v}, \overline{y})$ in $db'(S)$ and for all tuples $(\overline{v}'', \overline{w}'')$ in $db'(R)$, $\overline{v}'' \neq \overline{v}$. From this and the relational calculus semantics, it follows that $[\gamma]^{db} = \bot$. This is in contradiction with $[\gamma]^{db'} = \top$.

This completes the proof. $\square$

LEMMA F.15. *Let u be a user in $\mathcal{U}$, $P = \langle M, f^u_{conf}\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f^u_{conf} is as above, and L be the P-LTS. For any run $r \in traces(L)$ and any action $a \in \mathcal{A}_{D,u}$, if $extend(r, a)$ is defined, then a preserves the equivalence class for r, P, and u.*

PROOF. Let u be a user in $\mathcal{U}$, $P = \langle M, f^u_{conf}\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f^u_{conf} is as above, and L be the P-LTS. In the following, we use e to refer to the *extend* function and f to refer to f^u_{conf}. We prove our claim by contradiction. Assume, for contradiction's sake, that there is a run $r \in traces(L)$ and an action $a \in \mathcal{A}_{D,u}$ such that $e(r,a)$ is defined and a does not preserve the equivalence class for r, P, and u. According to the LTS semantics, the fact that $e(r,a)$ is defined implies that $triggers(last(r)) = \epsilon$. Therefore, $triggers(last(r')) = \epsilon$ holds as well for any for any $r' \in [\![r]\!]_{P,u}$ (because r and r' are indistinguishable and, therefore, their projections are consistent), and, thus, $e(r',a)$ is defined as well for any $r' \in [\![r]\!]_{P,u}$. There are a number of cases depending on a:

1. $a = \langle u, \texttt{SELECT}, q\rangle$. There are two cases:
   (a) $secEx(last(e(r,a))) = \bot$. From the LTS rules and $secEx(last(e(r,a))) = \bot$, it follows that $f(last(r),a) = \top$. From this and Lemma $F.9$, it follows that $f(last(r'),a) = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows $secEx(last(e(r',a))) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From $f(last(r'),a) = \top$ for any $r' \in [\![r]\!]_{P,u}$, it follows that $secure(u,q,last(r')) = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and Lemma $F.7$, it follows that $[q]^{last(r').db} = [q]^{last(r).db}$ for all $r' \in [\![r]\!]_{P,u}$. Furthermore, it follows trivially from the LTS rule ***SELECT** Success*, that the state after a's execution is data indistinguishable from $last(r)$. It is also easy to see that $e(r',a)$ is well-defined for any $r' \in [\![r]\!]_{P,u}$. From the considerations above and $r' \in [\![r]\!]_{P,u}$, it follows trivially that $e(r',a) \in [\![e(r,a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r',a)$. This leads to a contradiction.
   (b) $secEx(last(e(r,a))) = \top$. From the LTS rules and $secEx(last(e(r,a))) = \top$, it follows that $f(last(r),a) = \bot$. From this and Lemma $F.9$, it follows that $f(last(r'),a) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows $secEx(last(e(r',a))) = \top$ for any $r' \in [\![r]\!]_{P,u}$. The data indistinguishability between $last(e(r',a))$ and $last(e(r,a))$ follows trivially from the data indistinguishability between $last(r')$ and $last(r)$. Therefore, for any run $r' \in [\![r]\!]_{P,C}$, there is exactly one run $e(r',a)$. From the considerations above, it follows trivially that $e(r',a) \in [\![e(r,a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r',a)$. This leads to a contradiction.

   Both cases leads to a contradiction. This completes the proof for $a = \langle u, \texttt{SELECT}, q\rangle$.
2. $a = \langle u, \texttt{INSERT}, R, \bar{t}\rangle$. In the following, we denote by gI the function $getInfo$, by gS the function $getInfoS$, and by gV the function $getInfoV$. There are three cases:
   (a) $secEx(last(e(r,a))) = \bot$ and $Ex(last(e(r,a))) = \emptyset$. From the LTS rules and $secEx(last(e(r,a))) = \bot$, it follows that $f(last(r),a) = \top$. From this and Lemma $F.9$, it follows that $f(last(r'),a) = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows that $secEx(last(e(r',a))) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From f^u_{conf}'s definition and $f(last(r),a) = \top$, it follows that $secure(u, gS(\gamma, act), last(r))$ holds for any integrity constraint γ in $Dep(\Gamma, a)$. From $Ex(last(e(r,a))) = \emptyset$ and Proposition $F.7$, it follows $[gS(\gamma, act)]^{last(r).db} = \top$. From this, $secure(u, gS(\gamma, act), last(r))$, and Lemma $F.7$, it follows that $[gS(\gamma, act)]^{last(r').db} = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and Proposition $F.7$, it follows that $Ex(last(e(r', a))) = \emptyset$ for any $r' \in [\![r]\!]_{P,u}$. We claim that, for any $r' \in [\![r]\!]_{P,u}$, $last(e(r,a))$ and $last(e(r',a))$ are data indistinguishable. From this and the above considerations, it follows trivially that $e(r',a) \in [\![e(r,a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r',a)$. This leads to a contradiction.

   We now prove our claim that for any $r' \in [\![r]\!]_{P,u}$, $last(e(r,a))$ and $last(e(r',a))$ are data indistinguishable. We prove the claim by contradiction. Let $s_2 = \langle db_2, U_2, sec_2, T_2, V_2\rangle$ be $pState(last(e(r,a)))$, $s_2' = \langle db_2', U_2', sec_2', T_2', V_2'\rangle$ be $pState(last(e(r',a)))$, $s_1 = \langle db_1, U_1, sec_1, T_1, V_1\rangle$ be $pState(last(r))$, and $s_1' = \langle db_1', U_1', sec_1', T_1', V_1'\rangle$ be $pState(last(r'))$. In the following, we denote the *permissions* function by p. Furthermore, note that s_1 and s_1' are data-indistinguishable because $r' \in [\![r]\!]_{P,u}$. There are a number of cases:
   i. $U_2 \neq U_2'$. Since a is an `INSERT` operation, it follows that $U_1 = U_2$ and $U_1' = U_2'$. Furthermore, from $s_1 \cong^{data}_{u,M} s_1'$, it follows that $U_1 = U_1'$. Therefore, $U_2 = U_2'$ leading to a contradiction.
   ii. $sec_2 \neq sec_2'$. The proof is similar to the case $U_2 \neq U_2'$.
   iii. $T_2 \neq T_2'$. The proof is similar to the case $U_2 \neq U_2'$.
   iv. $V_2 \neq V_2'$. The proof is similar to the case $U_2 \neq U_2'$.
   v. there is a table R' for which $\langle \oplus, \texttt{SELECT}, R\rangle \in p(s_2,u)$ and $db_2(R') \neq db_2'(R')$. Note that $p(s_2,u) = p(s_1,u)$. There are two cases:
      - $R = R'$. From $s_1 \cong^{data}_{u,M} s_1'$ and $\langle \oplus, \texttt{SELECT}, R\rangle \in p(s_2,u)$, it follows that $db_1(R') = db_1'(R')$. From this and the fact that a has been executed successfully both in $e(r,a)$ and $e(r',a)$, it follows that $db_2(R') = db_1(R') \cup \{\bar{t}\}$ and $db_2'(R') = db_1'(R') \cup \{\bar{t}\}$. From this and $db_1(R') = db_1'(R')$, it follows that $db_2(R') = db_2'(R')$ leading to a contradiction.
      - $R \neq R'$. From $s_1 \cong^{data}_{P,u} s_1'$ and $\langle \oplus, \texttt{SELECT}, R\rangle \in p(s_2,u)$, it follows that $db_1(R') = db_1'(R')$. From this and the fact that a does not modify R', it follows that $db_1(R') = db_2(R')$ and $db_1'(R') = db_2'(R')$. From this and $db_1(R') = db_1'(R')$, it follows that $db_2(R') = db_2'(R')$ leading to a contradiction.
   vi. there is a view v for which $\langle \oplus, \texttt{SELECT}, v\rangle \in p(s_2,u)$ and $db_2(v) \neq db_2'(v)$. Note that $p(s_2, u) = p(s_1,u)$. Since a has been successfully executed in both states, we know that $noLeak(s_1, a, u)$ hold. There are two cases:
      - $R \notin tDet(v,s,M)$. Then, $v(s_1) = v(s_2)$ and $v(s_1') = v(s_2')$ (because R's content does not determine v's materialization). From $s_1 \cong^{data}_{u,M} s_1'$ and the fact that a modifies only R, it follows that $v(db_2) = v(db_2')$ leading to a contradiction.
      - $R \in tDet(v,s,M)$ and for all $o \in tDet(v, s, M)$, $\langle \oplus, \texttt{SELECT}, o\rangle \in p(s_1,u)$. From this and $s_1 \cong^{data}_{u,M} s_1'$, it follows that, for all $o \in tDet(v, s, M)$, $o(s_1) = o(s_1')$. If $o \neq R$, $o(s_1) = o(s_1') = o(s_2) = o(s_2')$. From $s_1 \cong^{data}_{u,M} s_1'$ and $\langle \oplus, \texttt{SELECT}, R\rangle \in p(s_1,u)$, it follows that $db_1(R) = db_1'(R)$. From this and the fact that a has been executed successfully both in $e(r,a)$ and $e(r',a)$, it follows that $db_2(R) = db_1(R) \cup \{\bar{t}\}$ and $db_2'(R) = db_1'(R) \cup \{\bar{t}\}$. From this and $db_1(R) = db_1'(R)$, it follows

that $db_2(R) = db'_2(R)$. From this and for all $o \in tDet(v, s, M)$ such that $o \neq R$, $o(s_2) = o(s'_2)$, it follows that for all $o \in tDet(v, s, M)$, $o(s_2) = o(s'_2)$. Since the content of all tables determining v is the same in s_2 and s'_2, it follows that $db_2(v) = db'_2(v)$ leading to a contradiction.

All the cases lead to a contradiction.

(b) $secEx(last(e(r, a))) = \bot$ and $Ex(last(e(r, a))) \neq \emptyset$. From the LTS rules and $secEx(e(r, a)) = \bot$, it follows that $f(last(r), a) = \top$. From this and Lemma F.9, it follows that $f(last(r'), a) = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows that $secEx(last(e(r', a))) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. Assume that the exception has been caused by the constraint γ, i.e., $\gamma \in Ex(last(e(r, a)))$. From this and Proposition F.7, it follows that $gV(\gamma, a)$ holds in $last(r).db$. From f^u_{conf}'s definition, it thus follows that $secure(u, gV(\gamma, a), last(r))$ holds. From this, $[gV(\gamma, a)]^{last(r).db} = \top$, and Lemma F.7, it follows that $[gV(\gamma, act)]^{last(r').db} = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and Proposition F.7, it follows that $\gamma \in Ex(last(e(r', a)))$ for any $r' \in [\![r]\!]_{P,u}$. The data indistinguishability between $last(e(r, a))$ and $last(e(r', a))$ follows trivially from the data indistinguishability between $last(r)$ and $last(r')$ for any $r' \in [\![r]\!]_{P,u}$. Therefore, for any run $r' \in [\![r]\!]_{P,u}$, there is exactly one run $e(r', a)$. From the considerations above, it follows trivially that $e(r', a) \in [\![e(r, a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r', a)$. This leads to a contradiction.

(c) $secEx(last(e(r, a))) = \top$. From the LTS rules and $secEx(last(e(r, a))) = \top$, it follows that $f(last(r), a) = \bot$. From this and Lemma F.9, it follows that $f(last(r'), a) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows $secEx(last(e(r', a))) = \top$ for any $r' \in [\![r]\!]_{P,u}$. The data indistinguishability between $last(e(r, a))$ and $last(e(r', a))$ follows trivially from the data indistinguishability between $last(r)$ and $last(r')$ for any $r' \in [\![r]\!]_{P,u}$. Therefore, for any run $r' \in [\![r]\!]_{P,u}$, there is exactly one run $e(r', a)$. From the considerations above, it follows trivially that $e(r', a) \in [\![e(r, a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r', a)$. This leads to a contradiction.

All cases lead to a contradiction. This completes the proof for $a = \langle u, \texttt{INSERT}, R, \bar{t}\rangle$.

3. $a = \langle u, \texttt{DELETE}, R, \bar{t}\rangle$. The proof is similar to that for $a = \langle u, \texttt{INSERT}, R, \bar{t}\rangle$.
4. $a = \langle \oplus, u', p, u\rangle$. There are two cases:

(a) $secEx(last(e(r, a))) = \bot$. We assume that $p = \langle \texttt{SELECT}, O\rangle$ for some $O \in D \cup V$. If this is not the case, the proof is trivial. Furthermore, we also assume that $u' = u$, otherwise the proof is, again, trivial since the new permission does not influence u's permissions. From the LTS rules and $secEx(last(e(r, a))) = \bot$, it follows that $f(last(r), a) = \top$. From this and Lemma F.9, it follows that $f(last(r'), a) = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows $secEx(last(e(r', a))) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From $secEx(last(e(r, a))) = \bot$ and f^u_{conf}'s definition, it follows that $last(r').sec = last(e(r', a)).sec$. Therefore, since $last(r)$ and $last(r')$ are data indistinguishable, for any $r' \in [\![r]\!]_{P,u}$, then also $last(e(r, a))$ and $last(e(r', a))$ are data indistinguishable. Therefore, for any run $r' \in [\![r]\!]_{P,u}$, there is exactly one run $e(r', a)$. From the considerations above, it follows trivially that $e(r', a) \in [\![e(r, a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r', a)$. This leads to a contradiction.

(b) $secEx(last(e(r, a))) = \top$. From the LTS rules and $secEx(last(e(r, a))) = \top$, it follows that $f(last(r), a) = \bot$. From this and Lemma F.9, it follows that $f(last(r'), a) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows $secEx(last(e(r', a))) = \top$ for any $r' \in [\![r]\!]_{P,u}$. The data indistinguishability between $last(e(r', a))$ and $last(e(r, a))$ follows trivially from the data indistinguishability between $last(r')$ and $last(r)$. Therefore, for any run $r' \in [\![r]\!]_{P,u}$, there is exactly one run $e(r', a)$. From the considerations above, it follows trivially that $e(r', a) \in [\![e(r, a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r', a)$. This leads to a contradiction.

Both cases lead to a contradiction. This completes the proof for $a = \langle \oplus, u', p, u\rangle$.

5. $a = \langle \oplus^*, u', p, u\rangle$. The proof is similar to that for $a = \langle \oplus, u', p, u\rangle$.
6. $a = \langle \ominus, u', p, u\rangle$. The proof is similar to that for $a = \langle u, \texttt{SELECT}, q\rangle$. The only difference is in proving that for any $r' \in [\![r]\!]_{P,u}$, $last(e(r, a))$ and $last(e(r', a))$ are data indistinguishable. Assume, for contradiction's sake, that this is not the case. Let $s_2 = \langle db_2, U_2, sec_2, T_2, V_2\rangle$ be $pState(last(e(r, a)))$, $s'_2 = \langle db'_2, U'_2, sec'_2, T'_2, V'_2\rangle$ be $pState(last(e(r', a)))$, $s_1 = \langle db_1, U_1, sec_1, T_1, V_1\rangle$ be $pState(last(r))$, and, finally, $s'_1 = \langle db'_1, U'_1, sec'_1, T'_1, V'_1\rangle$ be $pState(last(r'))$. In the following, we denote the *permissions* function by p. Furthermore, note that s_1 and s'_1 are data-indistinguishable because $r' \in [\![r]\!]_{P,u}$. There are a number of cases:

(a) $U_2 \neq U'_2$. Since a is an REVOKE operation, it follows that $U_1 = U_2$ and $U'_1 = U'_2$. Furthermore, from $s_1 \cong^{data}_{u,M} s'_1$, it follows that $U_1 = U'_1$. Therefore, $U_2 = U'_2$ leading to a contradiction.

(b) $sec_2 \neq sec'_2$. From $s_1 \cong^{data}_{u,M} s'_1$, it follows that $sec_1 = sec'_1$. From a's definition and the LTS rules, it follows that $sec_2 = revoke(sec_1, u', p, u)$ and $sec'_2 = revoke(sec'_1, u', p, u)$. From this and $sec_1 = sec'_1$, it follows that $sec_2 = sec'_2$ leading to a contradiction.

(c) $T_2 \neq T'_2$. The proof is similar to the case $U_2 \neq U'_2$.

(d) $V_2 \neq V'_2$. The proof is similar to the case $U_2 \neq U'_2$.

(e) there is a table R for which $\langle \oplus, \texttt{SELECT}, R\rangle \in p(s_2, u)$ and $db_2(R) \neq db'_2(R)$. Since a is an REVOKE operation, it follows that $db_1 = db_2$ and $db'_1 = db'_2$. Furthermore, from $s_1 \cong^{data}_{u,M} s'_1$, it follows that $db_1(R) = db'_1(R)$. From this, $db_1 = db_2$, and $db'_1 = db'_2$, it follows that $db_2(R) = db'_2(R)$ leading to a contradiction.

(f) there a view v for which $\langle \oplus, \texttt{SELECT}, v\rangle \in p(s_2, u)$ and $db_2(v) \neq db'_2(v)$. Since a is an REVOKE operation, it follows that $db_1 = db_2$ and $db'_1 = db'_2$. Furthermore, from $s_1 \cong^{data}_{u,M} s'_1$, it follows that $db_1(v) = db'_1(v)$. From this, $db_1 = db_2$, and $db'_1 = db'_2$, it follows that $db_2(v) = db'_2(v)$ leading to a contradiction.

All the cases lead to a contradiction.

7. $a = \langle u, \texttt{CREATE}, o\rangle$. The proof is similar to that for $a = \langle \ominus, u', p, u\rangle$.
8. $a = \langle u, \texttt{ADD\_USER}, u'\rangle$. The proof is similar to that for $a = \langle \ominus, u', p, u\rangle$.

This completes the proof. $\square$

LEMMA F.16. *Let u be a user in $\mathcal{U}$, $P = \langle M, f^u_{conf}\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f^u_{conf} is as above, and L be the P-LTS. For any run $r \in traces(L)$ such that $invoker(last(r)) = u$ and any trigger $t \in \mathcal{TRIGGER}_D$, if $extend(r, t)$ is defined, then t preserves the equivalence class for r, M, and u.*

PROOF. Let u be a user in $\mathcal{U}$, $P = \langle M, f^u_{conf}\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f^u_{conf} is as above, and L be the P-LTS. In the following, we use e to refer to the *extend* function. The proof in the cases where the trigger t is not enabled, i.e., its WHEN condition is not satisfied, or t's WHEN condition is not secure are similar to the proof of the SELECT case of Lemma $F.15$. In the following, we therefore assume that the trigger t is enabled and that its WHEN condition is secure. We prove our claim by contradiction. Assume, for contradiction's sake, that there is a run $r \in traces(L)$ such that $invoker(last(r)) = u$ and a trigger t such that $e(r, t)$ is defined and t does not preserve the equivalence class for r, P, and u. Since $invoker(last(r)) = u$ and $e(r, t)$ is defined, then $e(r', t)$ is defined as well for any $r' \in [\![r]\!]_{P,u}$ (indeed, from $invoker(last(r)) = u$, it follows that the last action in r is either an action issued by u or a trigger invoker by u. From this, the fact that $e(r, t)$ is defined, and the fact that r and r' are indistinguishable, it follows that $tr(last(r)) = tr(last(r')) = t$). Let a be t's action and $w = \langle u', \texttt{SELECT}, q\rangle$ be the SELECT command associated with t's WHEN condition. Let s be the state $last(r)$, s' be the state obtained just after the execution of the WHEN condition, and s'' be the state $last(e(r, t))$. There are a number of cases depending on t's action a:

1. $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$. There are three cases:
   (a) $secEx(s'') = \bot$ and $Ex(s'') = \emptyset$. The proof of this case is similar to that of the corresponding case in Lemma $F.15$.
   (b) $secEx(s'') = \bot$ and $Ex(s'') \neq \emptyset$. The only difference between the proof of this case in this Lemma and in that of Lemma $F.15$ is that we have to establish again the data indistinguishability between $last(e(r, t))$ and $last(e(r', t))$. Indeed, for triggers the roll-back state is, in general, different from the one immediately before the trigger's execution, i.e., it may be that $pState(last(e(r, t))) \neq pState(last(r))$. We now prove that $last(e(r, t))$ and $last(e(r', t))$ are data indistinguishable. From the LTS semantics, it follows that $r = p \cdot s_0 \cdot \langle invoker(last(r)), op, R', \overline{v}\rangle \cdot s_1 \cdot t_1 \cdot \ldots \cdot s_{n-1} \cdot t_n \cdot s_n$, where $p \in traces(L)$ and $t_1, \ldots, t_n \in \mathcal{TRIGGER}_D$. Similarly, $r' = p' \cdot s'_0 \cdot \langle invoker(last(r)), op, R', \overline{v}\rangle \cdot s'_1 \cdot t_1 \cdot \ldots \cdot s'_{n-1} \cdot t_n \cdot s'_n$, where $p' \in traces(L)$, $p \cong_{u,M} p'$, and all states s_i and s'_i are data indistinguishable. Then, the roll-back states are, respectively, s_0 and s'_0, which are data indistinguishable. From the LTS rules, $last(e(r, a)) = s_0$ and $last(e(r', a)) = s'_0$. Therefore, the data indistinguishability between $last(e(r, a))$ and $last(e(r', a))$ follows trivially for any $r' \in [\![r]\!]_{P,u}$.
   (c) $secEx(s'') = \top$. The proof is similar to the previous case.

   All cases lead to a contradiction. This completes the proof for $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$.
2. $a = \langle u', \texttt{DELETE}, R, \overline{t}\rangle$. The proof is similar to that for $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$.
3. $a = \langle \oplus, u'', p, u'\rangle$. There are two cases:
   (a) $secEx(s'') = \bot$. In this case, the proof is similar to the corresponding case in Lemma $F.15$.
   (b) $secEx(s'') = \top$. The proof is similar to the $secEx(s'') = \top$ case of the $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$ case.

   Both cases lead to a contradiction. This completes the proof for $a = \langle \oplus, u'', p, u'\rangle$.
4. $a = \langle \oplus^*, u'', p, u'\rangle$. The proof is similar to that for $a = \langle \oplus, u'', p, u'\rangle$.
5. $a = \langle \ominus, u'', p, u'\rangle$. The proof is similar to that for $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$.

This completes the proof. $\square$

We now prove our main result, namely that f^u_{conf} provides data confidentiality with respect to the user u. We first recall the concept of *derivation*. Given a judgment $r, i \vdash_u \phi$, a *derivation of $r, i \vdash_u \phi$ with respect to $\mathcal{ATK}_u$*, or *a derivation of $r, i \vdash_u \phi$* for short, is a proof tree, obtained by applying the rules defining $\mathcal{ATK}_u$, that ends in $r, i \vdash_u \phi$. With a slight abuse of notation, we use $r, i \vdash_u \phi$ to denote both the judgment and its derivation. The length of a derivation, denoted $|r, i \vdash_u \phi|$, is the number of rule applications in it.

THEOREM F.1. *Let u be a user in $\mathcal{U}$, $P = \langle M, f^u_{conf}\rangle$ be an extended configuration, where M is a system configuration and f^u_{conf} is as above. The PDP f^u_{conf} provides data confidentiality with respect to P, u, $\mathcal{ATK}_u$, and $\cong_{P,u}$.*

PROOF. Let u be a user in $\mathcal{U}$, $P = \langle M, f^u_{conf}\rangle$ be an extended configuration, where M is a system configuration and f^u_{conf} is as above, and L be the P-LTS. Furthermore, let r be a run in $traces(L)$, i be an integer such that $1 \leq i \leq |r|$, and ϕ be a sentence such that $r, i \vdash_u \phi$ holds. We claim that also $secure_{P,u}(r, i \vdash_u \phi)$ holds. The theorem follows trivially from the claim.

We now prove our claim that $secure_{P,u}(r, i \vdash_u \phi)$ holds. Let r be a run in $traces(L)$, i be an integer such that $1 \leq i \leq |r|$, and ϕ be a sentence such that $r, i \vdash_u \phi$ holds. Furthermore, in the following we use e to denote the *extend* function. We prove our claim by induction on the length of the derivation $r, i \vdash_u \phi$.

Base Case: Assume that $|r, i \vdash_u \phi| = 1$. There are a number of cases depending on the rule used to obtain $r, i \vdash_u \phi$.

1. *SELECT Success - 1.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{SELECT}, \phi\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = s'$, where $s' = \langle db, U, sec, T, V, c'\rangle$. From the rules, it follows that $f^u_{conf}(s', \langle u, \texttt{SELECT}, \phi\rangle) = \top$. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, s') = \top$ holds. From this, Lemma $F.8$, and $pState(s) = pState(s')$, it follows that $secure(u, \phi, s) = \top$ holds. From this, Lemma $F.7$, and $last(r^i) = s$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.
2. *SELECT Success - 2.* The proof for this case is similar to that of *SELECT Success - 1.*
3. *INSERT Success.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $R(\overline{t})$. Then,

$secure_{P,u}(r, i \vdash_u R(\overline{t}))$ holds. Indeed, in all runs r' indistinguishable from r^i the last action is $\langle u, \texttt{INSERT}, R, \overline{t}\rangle$. Furthermore, the action has been executed successfully. Therefore, according to the LTS rules, $\overline{t} \in db''(R)$, where $db'' = last(r').db$. From this and the relational calculus semantics, it follows that $[R(\overline{t})]^{last(r').db} = \top$. Therefore, $[R(\overline{t})]^{last(r').db} = \top$ for any run $r' \in [\![r^i]\!]_{P,u}$. Hence, $secure_{P,u}(r, i \vdash_u R(\overline{t}))$ holds.

4. *`INSERT` Success - FD.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $\neg\exists \overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f^u_{conf}(s', \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle) = \top$. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$ holds because ϕ is equivalent to $getInfoS(\gamma, a)$ for some $\gamma \in Dep(\Gamma, a)$, where $a = \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. We claim that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $F.2$ and $secure^{data}_{P,u}(r, i \vdash_u \phi)$, it follows $secure_{P,u}(r, i \vdash_u \phi)$.
We now prove our claim that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds. Let s' be the state $last(r^{i-1})$. Furthermore, for brevity's sake, in the following we omit the $pState$ function where needed. For instance, with a slight abuse of notation, we write $[\![s']\!]^{data}_{u,M}$ instead of $[\![pState(s')]\!]^{data}_{u,M}$. There are two cases:
   (a) the `INSERT` command has caused an integrity constraint violation, i.e., $Ex(s) \neq \emptyset$. From $secure(u, \phi, s') = \top$ and Lemma $F.7$, it follows that $secure^{data}_{P,u}(r, i-1 \vdash_u \phi)$ holds. From this, it follows that $[\phi]^v = [\phi]^{s'}$ for any $v \in [\![s']\!]^{data}_{u,M}$. From this and the fact that the `INSERT` command caused an exception (i.e., $s' = s$), it follows that $[\phi]^v = [\phi]^s$ for any $v \in [\![s]\!]^{data}_{u,M}$. From this, it follows that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds.
   (b) the `INSERT` command has not caused exceptions, i.e., $Ex(s) = \emptyset$. From $secure(u, \phi, s') = \top$ and Lemma $F.7$, it follows that $secure^{data}_{P,u}(r, i-1 \vdash_u \phi)$ holds. From this, it follows that $[\phi]^v = [\phi]^{s'}$ for any $v \in [\![s']\!]^{data}_{u,M}$. Furthermore, from Proposition $F.7$ and $Ex(s) = \emptyset$, it follows that ϕ holds in s'. Let $A_{s',R,\overline{t}}$ be the set $\{\langle db[R \oplus \overline{t}], U, sec, T, V\rangle \in \Pi_M \mid \exists db' \in \Omega_D.\, \langle db', U, sec, T, V\rangle \in [\![s']\!]^{data}_{u,M}\}$. It is easy to see that $[\![s]\!]^{data}_{u,M} \subseteq A_{s',R,\overline{t}}$. We now show that ϕ holds for any $z \in A_{s',R,\overline{t}}$. Let $z_1 \in [\![s']\!]^{data}_{u,M}$. From $[\phi]^v = [\phi]^{s'}$ for any $v \in [\![s']\!]^{data}_{u,M}$ and the fact that ϕ holds in s', it follows that $[\phi]^{z_1} = \top$. Therefore, for any $(\overline{k}_1, \overline{k}_2, \overline{k}_3) \in R(z_1)$ such that $|\overline{k}_1| = |\overline{v}|$, $|\overline{k}_2| = |\overline{w}|$, and $|\overline{k}_3| = |\overline{z}|$, if $k_1 = \overline{v}$, then $k_2 = \overline{w}$. Then, for any $(\overline{k}_1, \overline{k}_2, \overline{k}_3) \in R(z_1) \cup \{(\overline{v}, \overline{w}, \overline{q})\}$, if $k_1 = \overline{v}$, then $k_2 = \overline{w}$. Therefore, ϕ holds also in $z_1[R \oplus \overline{t}] \in A_{pState(s'),R,\overline{t}}$. Hence, $[\phi]^z = \top$ for any $z \in A_{s',R,\overline{t}}$. From this and $[\![s]\!]^{data}_{u,M} \subseteq A_{s',R,\overline{t}}$, it follows that $[\phi]^z = \top$ for any $z \in [\![s]\!]^{data}_{u,M}$. From this, it follows that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds.
5. *`INSERT` Success - ID.* The proof of this case is similar to that for the *`INSERT` Success - FD.*
6. *`DELETE` Success.* The proof for this case is similar to that of *`INSERT` Success.*
7. *`DELETE` Success - ID.* The proof of this case is similar to that for the *`INSERT` Success - FD.*
8. *`INSERT` Exception.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSER}, R, \overline{t}\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $\neg R(\overline{t})$. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f^u_{conf}(s', \langle u, \texttt{INSERT}, R, \overline{t}\rangle) = \top$. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$ holds because $\phi = getInfo(\langle u, \texttt{INSERT}, R, \overline{t}\rangle)$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From the LTS semantics, it follows that $pState(s) \cong^{data}_{u,M} pState(last(r^{i-1}))$. From this, Lemma $F.8$, and $secure(u, \phi, last(r^{i-1})) = \top$, it follows that $secure(u, \phi, last(r^i)) = \top$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.
9. *`DELETE` Exception.* The proof for this case is similar to that of *`INSERT` Exception.*
10. *`INSERT` FD Exception.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $\exists \overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f^u_{conf}(s', \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle) = \top$. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$ because $\phi = getInfoV(\gamma, \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle)$ for some constraint $\gamma \in Dep(\Gamma, \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle)$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From the LTS semantics, it follows that $pState(s) \cong^{data}_{u,M} pState(last(r^{i-1}))$. From this, $secure(u, \phi, last(r^{i-1})) = \top$, and Lemma $F.8$, it follows that $secure(u, \phi, last(r^i)) = \top$. From this and Lemma $F.7$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
11. *`INSERT` ID Exception.* The proof for this case is similar to that of *`INSERT` FD Exception.*
12. *`DELETE` FD Exception.* The proof for this case is similar to that of *`INSERT` FD Exception.*
13. *Integrity Constraint.* The proof of this case follows trivially from the fact that for any state $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and any $\gamma \in \Gamma$, $[\gamma]^{db} = \top$ holds by definition.
14. *Learn `GRANT/REVOKE` Backward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec', T, V, c'\rangle$, and t be a trigger whose `WHEN` condition is ϕ and whose action is either a `GRANT` or a `REVOKE`. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f^u_{conf}(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$, where u' is either the trigger's owner or the trigger's invoker depending on the security mode. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$. From this and F.7, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.
15. *Trigger `GRANT` Disabled Backward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec', T, V, c'\rangle$, and t be a trigger whose `WHEN` condition is ψ, and ϕ be $\neg\psi$. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f^u_{conf}(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$, where u' is either the trigger's owner or the trigger's invoker depending on the se-

curity mode. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.

16. *Trigger* ***REVOKE*** *Disabled Backward.* The proof for this case is similar to that of *Trigger* ***GRANT*** *Disabled Backward.*
17. *Trigger* ***INSERT*** *FD Exception.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec', T, V, c'\rangle$, and t be a trigger whose `WHEN` condition is ϕ and whose action act is a `INSERT` statement $\langle u', \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle$. Furthermore, let ϕ be $\exists \overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f^u_{conf}(last(r^{i-1}), act) = \top$. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$ because $\phi = getInfoV(\gamma, act)$ for some constraint $\gamma \in Dep(\Gamma, act)$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.
18. *Trigger* ***INSERT*** *ID Exception.* The proof for this case is similar to that of *Trigger* ***INSERT*** *ID Exception.*
19. *Trigger* ***DELETE*** *ID Exception.* The proof for this case is similar to that of *Trigger* ***DELETE*** *ID Exception.*
20. *Trigger Exception.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec', T, V, c'\rangle$, and t be a trigger whose `WHEN` condition is ϕ and whose action is act. From the rule's definition, it follows $f^u_{conf}(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$, where u' is either the trigger's owner or the trigger's invoker depending on the security mode. From this and f^u_{conf}'s definition, it follows $secure(u, \phi, last(r^{i-1})) = \top$. From this and F.7, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.
21. *Trigger* ***INSERT*** *Exception.* The proof for this case is similar to that of ***INSERT*** *Exception.*
22. *Trigger* ***DELETE*** *Exception.* The proof for this case is similar to that of ***DELETE*** *Exception.*
23. *Trigger Rollback* ***INSERT***. Let i be such that $r^i = r^{i-n-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s_1 \cdot t_1 \cdot s_2 \cdot \ldots \cdot t_n \cdot s_n$, where $s_1, s_2, \ldots, s_n \in \Omega_M$ and $t_1, \ldots, t_n \in \mathcal{TRIGGER}_D$, and ϕ be $\neg R(\overline{t})$. Furthermore, let $last(r^{i-n-1}) = \langle db', U', sec', T', V', c'\rangle$ and s_n be $\langle db, U, sec, T, V, c\rangle$. From the rule's definition, it follows that $secEx(s_1) = \bot$. From this, it follows that $f^u_{conf}(last(r^{i-n-1}), \langle u, \texttt{INSERT}, R, \overline{t}\rangle) = \top$. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-n-1})) = \top$ because $\phi = getInfo(\langle u, \texttt{INSERT}, R, \overline{t}\rangle)$. From the LTS semantics, it follows that $last(r^{i-n-1}) \cong^{data}_{u,M} s_n$. From this, $secure(u, \phi, last(r^{i-n-1})) = \top$, and Lemma $F.8$, it follows that $secure(u, \phi, s_n) = \top$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.
24. *Trigger Rollback* ***DELETE***. The proof for this case is similar to that of *Trigger Rollback* ***INSERT***.

This completes the proof of the base step.

Induction Step: Assume that the claim hold for any derivation of $r, j \vdash_u \psi$ such that $|r, j \vdash_u \psi| < |r, i \vdash_u \phi|$. We now prove that the claim also holds for $r, i \vdash_u \phi$. There are a number of cases depending on the rule used to obtain $r, i \vdash_u \phi$.

1. *View.* The proof of this case follows trivially from the semantics of the relational calculus extended over views.
2. *Propagate Forward* ***SELECT***. Let i be such that $r^{i+1} = r^i \cdot \langle u, \texttt{SELECT}, \psi\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^i) = \langle db', U', sec', T', V', c'\rangle$. From the rule, it follows that $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $F.15$, the action $\langle u, \texttt{SELECT}, \psi\rangle$ preserves the equivalence class with respect to r^i, P, and u. From this, Lemma $F.12$, and $secure_{P,u}(r, i \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i+1 \vdash_u \phi)$ holds.
3. *Propagate Forward* ***GRANT/REVOKE***. Let i be such that $r^{i+1} = r^i \cdot \langle op, u', p, u\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^i) = \langle db', U', sec', T', V', c'\rangle$. From the rule, it follows that $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $F.15$, the action $\langle op, u', p, u\rangle$ preserves the equivalence class with respect to r^i, P, and u. From this, Lemma $F.13$, and $secure_{P,u}(r, i \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i+1 \vdash_u \phi)$ holds.
4. *Propagate Forward* ***CREATE***. The proof for this case is similar to that of *Propagate Forward* ***SELECT***.
5. *Propagate Backward* ***SELECT***. Let i be such that $r^{i+1} = r^i \cdot \langle u, \texttt{SELECT}, \psi\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$ and $last(r^i) = \langle db, U, sec, T, V, c\rangle$. From the rule, it follows that $r, i+1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i+1 \vdash_u \phi)$ holds. From Lemma $F.15$, the action $\langle u, \texttt{SELECT}, \psi\rangle$ preserves the equivalence class with respect to r^i, P, and u. From this, Lemma $F.12$, and $secure_{P,u}(r, i+1 \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
6. *Propagate Backward* ***GRANT/REVOKE***. Let i be such that $r^{i+1} = r^i \cdot \langle op, u', p, u\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$ and $last(r^i) = \langle db, U, sec, T, V, c\rangle$. From the rule, it follows that $r, i+1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i+1 \vdash_u \phi)$ holds. From Lemma $F.15$, the action $\langle op, u', p, u\rangle$ preserves the equivalence class with respect to r^i, P, and u. From this, Lemma $F.13$, and $secure_{P,u}(r, i+1 \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
7. *Propagate Backward* ***CREATE TRIGGER***. The proof for this case is similar to that of *Propagate Backward* ***SELECT***.
8. *Propagate Backward* ***CREATE VIEW***. Note that the formulae ψ and $replace(\psi, o)$ are semantically equivalent. This is the only difference between the proof for this case and the one for the *Propagate Backward* ***SELECT*** case.
9. *Rollback Backward - 1.* Let i be such that $r^i = r^{i-n-1} \cdot \langle u, op, R, \overline{t}\rangle \cdot s_1 \cdot t_1 \cdot s_2 \cdot \ldots \cdot t_n \cdot s_n$, where $s_1, s_2, \ldots, s_n \in \Omega_M$, $t_1, \ldots, t_n \in \mathcal{TRIGGER}_D$, and op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$. Furthermore, let s_n be $\langle db', U', sec', T', V', c'\rangle$ and $last(r^{i-n-1})$ be $\langle db, U, sec, T, V, c\rangle$. From the rule's definition, $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $F.16$, the triggers t_j preserve the equivalence class with respect to $r^{i-n-1+j}$, P, and u for any $1 \leq j \leq n$. Therefore, for any $v \in [\![r^{i-1}]\!]_{P,u}$, the run $e(v, t_n)$ contains the roll-back. Therefore, for any $v \in [\![r^{i-1}]\!]_{P,u}$, the state $last(e(v, t_n))$ is the state just before the action $\langle u, op, R, \overline{t}\rangle$. Let A be the set of partial states associated with the roll-back states. It is easy to see that A is the same as $\{pState(last(t')) | t' \in [\![r^{i-n-1}]\!]_{P,u}\}$. From $secure_{P,u}(r, i \vdash_u \phi)$, it follows that ϕ has the same result over all states in A. From this and

$A = \{pState(last(t'))|t' \in [\![r^{i-n-1}]\!]_{P,u}\}$, it follows that ϕ has the same result over all states in $\{pState(last(t'))| t' \in [\![r^{i-n-1}]\!]_{P,u}\}$. From this, it follows that $secure_{P,u}(r, i-n-1 \vdash_u \phi)$ holds.

10. *Rollback Backward - 2.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$. From the rule's definition, $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $F.15$, the action $\langle u, op, R, \bar{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. From this, Lemma $F.11$, the fact that the action does not modify the database state, and $secure_{P,u}(r, i \vdash_u \phi)$, it follows $secure_{P,u}(r, i-1 \vdash_u \phi)$.
11. *Rollback Forward - 1.* Let i be such that $r^i = r^{i-n-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s_1 \cdot t_1 \cdot s_2 \cdot \ldots \cdot t_n \cdot s_n$, where $s_1, s_2, \ldots, s_n \in \Omega_M$, $t_1, \ldots, t_n \in \mathcal{TRIGGER}_D$, and op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$. Furthermore, let s_n be $\langle db, U, sec, T, V, c\rangle$ and $last(r^{i-n-1})$ be $\langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-n-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-n-1 \vdash_u \phi)$ holds. From Lemma $F.16$, the triggers t_j preserve the equivalence class with respect to $r^{i-n-1+j}$, P, and u for any $1 \leq j \leq n$. Independently on the cause of the roll-back (either a security exception or an integrity constraint violation), we claim that the set A of roll-back partial states is $\{pState(last(t'))|t' \in [\![r^{i-n-1}]\!]_{P,u}\}$. From $secure_{P,u}(r, i-n-1 \vdash_u \phi)$, the result of ϕ is the same for all states in A. From this and $A = \{pState(last(t'))|t' \in [\![r^{i-n-1}]\!]_{P,u}\}$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
We now prove our claim. It is trivial to see (from the LTS's semantics) that the set of rollback's states is a subset of $\{pState(last(v))|v \in [\![r^{i-n-1}]\!]_{P,u}\}$. Assume, for contradiction's sake, that there is a state in $\{pState(last(v))|v \in [\![r^{i-n-1}]\!]_{P,u}\}$ that is not a rollback state for the runs in $[\![r^i]\!]_{P,u}$. This is impossible since all triggers $t_1, \ldots, t_n$ preserve the equivalence class.
12. *Rollback Forward - 2.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, where $op \in \{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From Lemma $F.15$, the action $\langle u, op, R, \bar{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. From this, Lemma $F.11$, the fact that the action does not modify the database state, and $secure_{P,u}(r, i-1 \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
13. *Propagate Forward* ***INSERT/DELETE*** *Success.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, where $op \in \{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From Lemma $F.15$, the action $\langle u, op, R, \bar{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. From $reviseBelif(r^{i-1}, \phi, r^i)$, it follows that the execution of $\langle u, op, R, \bar{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ for any $v \in [\![r^{i-1}]\!]_{P,u}$. From this, Lemma $F.11$, and $secure_{P,u}(r, i-1 \vdash_u \phi)$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.
14. *Propagate Forward* ***INSERT*** *Success - 1.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, where op is one if $\{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From Lemma $F.15$, the action $\langle u, op, R, \bar{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. We claim that the execution of $\langle u, \texttt{INSERT}, R, \bar{t}\rangle$ does not alter the content of the tables in $tables(\phi)$. From this, $secure_{P,u}(r, i-1 \vdash_u \phi)$, and Lemma $F.11$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.
We now prove our claim that the execution of $\langle u, \texttt{INSERT}, R, \bar{t}\rangle$ does not alter the content of the tables in $tables(\phi)$. From the rule's definition, it follows that $r, i-1 \vdash_u R(\bar{t})$ holds. From this and Lemma $B.1$, it follows that $[R(\bar{t})]^{last(r^{i-1}).db} = \top$. From $r, i-1 \vdash_u R(\bar{t})$ and the induction hypothesis, it follows that $secure_{P,u}(r, i-1, u, R(\bar{t}))$ holds. From this and $[R(\bar{t})]^{last(r^{i-1}).db} = \top$, it follows that $[R(\bar{t})]^{last(v).db} = \top$ for any $v \in [\![r^{i-1}]\!]_{P,u}$. From this and the relational calculus semantics, it follows that the execution of $\langle u, op, R, \bar{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ for any $v \in [\![r^{i-1}]\!]_{P,u}$.
15. *Propagate Forward* ***DELETE*** *Success - 1.* The proof for this case is similar to that of *Propagate Forward* ***INSERT*** *Success - 1.*
16. *Propagate Backward* ***INSERT/DELETE*** *Success.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, where $op \in \{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $F.15$, the action $\langle u, op, R, \bar{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. From $reviseBelif(r^{i-1}, \phi, r^i)$, it follows that the execution of $\langle u, op, R, \bar{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ for any $v \in [\![r^{i-1}]\!]_{P,u}$. From this, Lemma $F.11$, and $secure_{P,u}(r, i \vdash_u \phi)$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.
17. *Propagate Backward* ***INSERT*** *Success - 1.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, where op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $F.15$, the action $\langle u, op, R, \bar{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. We claim that the execution of $\langle u, \texttt{INSERT}, R, \bar{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ for any $v \in [\![r^{i-1}]\!]_{P,u}$ (the proof of this claim is in the proof of the *Propagate Forward* ***INSERT*** *Success - 1* case). From this, Lemma $F.11$, and $secure_{P,u}(r, i \vdash_u \phi)$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.
18. *Propagate Backward* ***DELETE*** *Success - 1.* The proof for this case is similar to that of *Propagate Forward* ***DELETE*** *Success - 1.*
19. *Reasoning.* Let Δ be a subset of $\{\delta \,|\, r, i \vdash_u \delta\}$ and $last(r^i) = \langle db, U, sec, T, V, c\rangle$. From the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \delta)$ holds for any $\delta \in \Delta$. Note that, given any $\delta \in \Delta$, from $r, i \vdash_u \delta$ and Lemma $B.1$, it follows that δ holds in $last(r^i)$. From this, $secure_{P,u}(r, i \vdash_u \delta)$ holds for any $\delta \in \Delta$,

$\Delta \models_{fin} \phi$, and Lemma $F.10$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.

20. *Learn* `INSERT` *Backward - 3.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and ϕ be $\neg R(\overline{t})$. From the rule's definition, $secEx(s) = \bot$. From this and the LTS rules, it follows that $f^u_{conf}(last(r^{i-1}), \langle u, \texttt{INSERT}, R, \overline{t}\rangle) = \top$. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$ because $\phi = getInfo(\langle u, \texttt{INSERT}, R, \overline{t}\rangle)$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.
21. *Learn* `DELETE` *Backward - 3.* The proof for this case is similar to that of *Learn* `INSERT` *Backward - 3*.
22. *Propagate Forward Disabled Trigger.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and t be a trigger. Furthermore, let ψ be t's condition where all free variables are replaced with $tpl(last(r^{i-1}))$. From the rule, it follows that $r, i-1 \vdash_u \phi$. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. Furthermore, from Lemma $G.8$, it follows that t preserves the equivalence class with respect to r^{i-1}, P, and u. If the trigger's action is an `INSERT` or a `DELETE` operation, we claim that the operation does not change the content of any table in $tables(\phi)$ for any run $v \in [\![r^{i-1}]\!]_{P,u}$. From this, the fact that t preserves the equivalence class with respect to r^{i-1}, P, and u, Lemma $F.14$, and $secure_{P,u}(r, i-1 \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
We now prove our claim. Assume that t's action in either an `INSERT` or a `DELETE` operation. From the rule, it follows that $r, i-1 \vdash_u \neg\psi$. From this and Lemma $B.1$, $[\psi]^{last(r^{i-1})} = \bot$. From $r, i-1 \vdash_u \neg\psi$ and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \psi)$ holds. From this and $[\psi]^{last(r^{i-1}).db} = \bot$, it follows that $[\psi]^{v.db} = \bot$ for any run $v \in [\![r^{i-1}]\!]_{P,u}$. Therefore, the trigger t is disabled in any run $v \in [\![r^{i-1}]\!]_{P,u}$. From this and the LTS semantics, it follows that t's execution does not change the content of any table in $tables(\phi)$ for any run $v \in [\![r^{i-1}]\!]_{P,u}$.
23. *Propagate Backward Disabled Trigger.* The proof for this case is similar to that of *Propagate Forward Disabled Trigger*.
24. *Learn* `INSERT` *Forward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and t be a trigger, and ϕ be $R(\overline{t})$. Furthermore, let ψ be t's condition where all free variables are replaced with $tpl(last(r^{i-1}))$. From the rule's definition, it follows that t's action is $\langle u', \texttt{INSERT}, R, \overline{t}\rangle$ and that $r, i-1 \vdash_u \psi$ holds. From Lemma $B.1$ and $r, i-1 \vdash_u \psi$, it follows that $[\psi]^{last(r^{i-1}).db} = \top$. From this, $secEx(s) = \bot$, and $Ex(s) = \emptyset$, it follows that t's action has been executed successfully. From this, it follows that $\overline{t} \in s.db(R)$. From $r, i-1 \vdash_u \psi$ and the induction hypothesis, it follows $secure_{P,u}(r, i-1 \vdash_u \psi)$. From this and $[\psi]^{last(r^{i-1}).db} = \top$, it follows that $[\psi]^{last(v).db} = \top$ for any $v \in [\![r^{i-1}]\!]_{P,u}$. From this, it follows that the trigger t is enabled in any run $v \in [\![r^{i-1}]\!]_{P,u}$. From Lemma $F.16$, it follows that t preserves the equivalence class with respect to r^{i-1}, P, and u. From this, $secEx(s) = \bot$, $Ex(s) = \emptyset$, and the fact that the trigger t is enabled in any run $v \in [\![r^{i-1}]\!]_{P,u}$, it follows that t's action is executed successfully in any run $e(v, t)$, where $v \in [\![r^{i-1}]\!]_{P,u}$. From this, it follows that $\overline{t} \in db''(R)$ for any $v \in [\![r^{i-1}]\!]_{P,u}$, where $db'' = last(e(v, t)).db$. Therefore, $secure_{P,u}(r, i \vdash_u \phi)$ holds.
25. *Learn* `INSERT` *- FD.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$, and $t \in \mathcal{TRIGGER}_D$, and ϕ be $\neg\exists \overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. Furthermore, let ψ be t's condition where all free variables are replaced with the values in $tpl(last(r^{i-1}))$ and $\langle u', \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle$ be t's actual action. From the rule, it follows that $r, i-1 \vdash_u \psi$. From this and Lemma $B.1$, it follows that $[\psi]^{last(r^{i-1}).db} = \top$. From this, $Ex(s) = \emptyset$, and $secEx(s) = \bot$, it follows that $f^u_{conf}(s', \langle u', \texttt{INSERT}, R, \overline{t}\rangle) = \top$, where s' is the state just after the execution of the `SELECT` statement associated with t's `WHEN` clause. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, s') = \top$. From this, $pState(s') = pState(last(r^{i-1}))$, and Lemma $F.8$, it follows that $secure(u, \phi, last(r^{i-1})) = \top$. From this and Lemma $F.7$, it follows also that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. We claim that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds. From this and Lemma $F.2$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
We now prove our claim that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds. Let s' be the state just after the execution of the `SELECT` statement associated with t's `WHEN` clause and s'' be the state $last(r^{i-1})$. Furthermore, for brevity's sake, in the following we omit the $pState$ function where needed. For instance, with a slight abuse of notation, we write $[\![s']\!]^{data}_{u,M}$ instead of $[\![pState(s')]\!]^{data}_{u,M}$. From $secure(u, \phi, s') = \top$, $s' \cong^{data}_{u,M} s''$, Lemma $F.8$, and Lemma $F.7$, it follows that $secure^{data}_{P,u}(r, i-1 \vdash_u \phi)$ holds. From this, it follows that $[\phi]^v = [\phi]^{s''}$ for any $v \in [\![s'']\!]^{data}_{u,M}$. Furthermore, from Proposition $F.7$ and $Ex(s) = \emptyset$, it follows that ϕ holds in s''. Let $A_{s'',R,\overline{t}}$ be the set $\{\langle db[R \oplus \overline{t}], U, sec, T, V\rangle \in \Pi_M \mid \exists db' \in \Omega_D.\, \langle db', U, sec, T, V\rangle \in [\![s'']\!]^{data}_{u,M}\}$. It is easy to see that $[\![s]\!]^{data}_{u,M} \subseteq A_{s'',R,\overline{t}}$. We now show that ϕ holds for any $z \in A_{s'',R,\overline{t}}$. Let $z_1 \in [\![s'']\!]^{data}_{u,M}$. From $[\phi]^v = [\phi]^{s''}$ for any $v \in [\![s'']\!]^{data}_{u,M}$ and the fact that ϕ holds in s'', it follows that $[\phi]^{z_1} = \top$. Therefore, for any $(\overline{k}_1, \overline{k}_2, \overline{k}_3) \in R(z_1)$, if $k_1 = \overline{v}$, then $k_2 = \overline{w}$. Then, for any $(\overline{k}_1, \overline{k}_2, \overline{k}_3) \in R(z_1) \cup \{(\overline{v}, \overline{w}, \overline{q})\}$, if $k_1 = \overline{v}$, then $k_2 = \overline{w}$. Therefore, ϕ holds also in $z_1[R \oplus \overline{t}] \in A_{pState(s''),R,\overline{t}}$. Hence, $[\phi]^z = \top$ for any $z \in A_{s'',R,\overline{t}}$. From this and $[\![s]\!]^{data}_{u,M} \subseteq A_{s'',R,\overline{t}}$, it follows that $[\phi]^z = \top$ for any $z \in [\![s]\!]^{data}_{u,M}$. From this, it follows that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds.
26. *Learn* `INSERT` *- FD - 1.* The proof of this case is similar to that of *Learn* `INSERT` *- FD*.
27. *Learn* `INSERT` *- ID.* The proof of this case is similar to that of *Learn* `INSERT` *- FD*. See also the proof of `INSERT` *Success - ID*.
28. *Learn* `INSERT` *- ID - 1.* The proof of this case is similar to that of *Learn* `INSERT` *- ID*.
29. *Learn* `INSERT` *Backward - 1.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and $t \in \mathcal{TRIGGER}_D$, and ϕ be t's actual `WHEN` condition, where all free variables are replaced with the values in $tpl(last(r^{i-1}))$. From the rule's definition, it follows that $secEx(s) = \top$. From this, the LTS semantics, and $secEx(s) =$

$\top$, it follows that $f^u_{conf}(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$. From this and Lemma $F.7$, it follows that also $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.

30. *Learn* `INSERT` *Backward - 2.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and $t \in \mathcal{TRIGGER}_D$, and ϕ be $\neg R(\bar{t})$. Furthermore, let $act = \langle u', \texttt{INSERT}, R, \bar{t}\rangle$ be t's actual action and γ be t's actual `WHEN` condition obtained by replacing all free variables with the values in $tpl(last(r^{i-1}))$. From the rule's definition, it follows that $secEx(s) = \top$ and there is a ψ such that $r, i-1 \vdash_u \psi$ and $r, i \vdash_u \neg\psi$. We claim that $[\gamma]^{db} = \top$. From this and $secEx(s) = \top$, it follows $f^u_{conf}(s', \langle u', \texttt{INSERT}, R, \bar{t}\rangle) = \top$, where s' is the state obtained after the evaluation of t's `WHEN` condition. From this and f^u_{conf}'s definition, it follows $secure(u, \phi, s') = \top$ since ϕ is equivalent to $getInfo(\langle u', \texttt{INSERT}, R, \bar{t}\rangle)$. From this, $pState(last(r^{i-1})) = pState(s')$, and Lemma $F.8$, it follows that $secure(u, \phi, last(r^{i-1})) = \top$. From this and Lemma $F.7$, it follows $secure_{P,u}(r, i-1 \vdash_u \phi)$.
We now prove our claim that $[\gamma]^{db} = \top$. Assume, for contradiction's sake, that this is not the case. From this and the LTS rules, it follows that $db = db'$. From the rule's definition, it follows that there is a ψ such that $r, i-1 \vdash_u \psi$ and $r, i \vdash_u \neg\psi$. From this, Lemma $B.1$, $s = \langle db', U', sec', T', V', c'\rangle$, and $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, it follows that $[\psi]^{db} = \top$ and $[\neg\psi]^{db'} = \top$. Therefore, $[\psi]^{db} = \top$ and $[\psi]^{db'} = \bot$. Hence, $db \neq db'$, which contradicts $db = db'$.
31. *Learn* `DELETE` *Forward.* The proof of this case is similar to that of *Learn* `INSERT` *Forward.*
32. *Learn* `DELETE` *- ID.* The proof of this case is similar to that of *Learn* `INSERT` *- FD.* See also the proof of `DELETE` *Success - ID.*
33. *Learn* `DELETE` *- ID - 1.* The proof of this case is similar to that of *Learn* `DELETE` *- ID.*
34. *Learn* `DELETE` *Backward - 1.* The proof of this case is similar to that of *Learn* `INSERT` *Backward - 1.*
35. *Learn* `DELETE` *Backward - 2.* The proof of this case is similar to that of *Learn* `INSERT` *Backward - 2.*
36. *Propagate Forward Trigger Action.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where t is a trigger, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From Lemma $F.16$, the trigger t preserves the equivalence class with respect to r^{i-1}, P, and u. We claim that the execution of t does not alter the content of the tables in $tables(\phi)$. From this, Lemma $F.11$, and $secure_{P,u}(r, i-1 \vdash_u \phi)$, it follows that also the judgment $r, i \vdash_u \phi$ is secure, i.e., $secure_{P,u}(r, i \vdash_u \phi)$ holds.
We now prove our claim that the execution of t does not alter the content of the tables in $tables(\phi)$. If the trigger is not enabled, proving the claim is trivial. In the following, we assume the trigger is enabled. There are four cases:
    - t's action is an `INSERT` statement. This case amount to claiming that the `INSERT` statement $\langle u', \texttt{INSERT}, R, \bar{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ in case $reviseBelif(r^{i-1}, \phi, r^i) = \top$. We proved the claim above in the *Propagate Forward* `INSERT/DELETE` *Success* case.
    - t's action is an `DELETE` statement. The proof is similar to that of the `INSERT` case.
    - t's action is an `GRANT` statement. In this case, the action does not alter the database state and the claim follows trivially.
    - t's action is an `REVOKE` statement. The proof is similar to that of the `GRANT` case.
37. *Propagate Backward Trigger Action.* The proof of this case is similar to *Propagate Backward Trigger Action.*
38. *Propagate Forward* `INSERT` *Trigger Action.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where t is a trigger, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From Lemma $F.16$, the trigger t preserves the equivalence class with respect to r^{i-1}, P, and u. We claim that the execution of t does not alter the content of the tables in $tables(\phi)$. From this, Lemma $F.11$, and $secure_{P,u}(r, i-1 \vdash_u \phi)$, it follows that also the judgment $r, i \vdash_u \phi$ is secure, i.e., $secure_{P,u}(r, i \vdash_u \phi)$ holds.
We now prove our claim that the execution of t does not alter the content of the tables in $tables(\phi)$. If the trigger is not enabled, proving the claim is trivial. In the following, we assume the trigger is enabled. Then, t's action is an `INSERT` statement. This case amount to claiming that the `INSERT` statement $\langle u', \texttt{INSERT}, R, \bar{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ in case $r, i-1 \vdash_u R(\bar{t})$ holds. We proved the claim above in the *Propagate Forward* `INSERT` *Success - 1* case.
39. *Propagate Forward* `DELETE` *Trigger Action.* The proof of this case is similar to that of *Propagate Forward* `INSERT` *Trigger Action.*
40. *Propagate Backward* `INSERT` *Trigger Action.* The proof of this case is similar to that of *Propagate Forward* `INSERT` *Trigger Action.*
41. *Propagate Backward* `DELETE` *Trigger Action.* The proof of this case is similar to that of *Propagate Forward* `INSERT` *Trigger Action.*
42. *Trigger FD* `INSERT` *Disabled Backward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$, $t \in \mathcal{TRIGGER}_D$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and ψ be t's actual `WHEN` condition obtained by replacing all free variables with the values in $tpl(last(r^{i-1}))$. Furthermore, let $act = \langle u', \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle$ be t's actual action and α be $\exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. From the rule's definition, it follows that $secEx(s) = \bot$. From this, it follows that $f^u_{conf}(last(r^{i-1}), \langle u', \texttt{SELECT}, \psi\rangle) = \top$. From this and f^u_{conf}'s definition, it follows that $secure(u, \neg\psi, last(r^{i-1})) = \top$. From this, it follows $secure(u, \psi, last(r^{i-1})) = \top$. From this and Lemma $F.7$, it follows $secure_{P,u}(r, i-1 \vdash_u \psi)$.
43. *Trigger ID* `INSERT` *Disabled Backward.* The proof of this case is similar to that of *Trigger FD* `INSERT` *Disabled Backward.*
44. *Trigger ID* `DELETE` *Disabled Backward.* The proof of this case is similar to that of *Trigger FD* `INSERT` *Disabled Backward.*

This completes the proof of the induction step.
This completes the proof. $\square$

F.3 Complexity proofs

In this section, we prove that data complexity of f^u_{conf} is AC^0. Note that the complexity class AC^0 identifies those problems that can be solved using constant-depth, polynomial-size boolean circuits with AND, OR, and NOT gates with unbounded fan-in. Note also that, in the following, with AC^0 we usually refer to *uniform-*AC^0. Given a database schema D and a database state $db \in \Omega^\Gamma_D$, the *size of* db, denoted also as $|db|$, is $|db| = \Sigma_{R\in D}\Sigma_{\overline{t}\in db(R)}|\overline{t}|$, where the size $|\overline{t}|$ of a tuple $\overline{t}$ is just its cardinality. Similarly, the *the size of the schema* D, denoted $|D|$, is $\Sigma_{R\in D}|R|$. Finally, given a set of views V over D, the *size of the extended vocabulary* $extVocabulary(D, V)$, denoted $|extVoc(D, V)|$, is $\Sigma_{o\in R\cup V}\Sigma_{0\le i<|o|}\dfrac{|o|!}{(|o|-i)!\cdot i!}$. Note that, given a view V, we denote by $|V|$ its cardinality. Furthermore, given a RC-formula ϕ, the *size of* ϕ, denoted as $|\phi|$, is defined as follows:

$$|\phi| = \begin{cases} 1+|\overline{x}| & \text{if } \phi := R(\overline{x}) \\ 1 & \text{if } \phi := \top \\ 1 & \text{if } \phi := \bot \\ 3 & \text{if } \phi := x = y \\ 1+|\psi|+|\gamma| & \text{if } \phi := \psi\ O\ \gamma \text{ and } O \in \{\vee,\wedge\} \\ 1+|\psi| & \text{if } \phi := \neg\psi \\ 2+|\psi| & \text{if } \phi := Q\,x.\,\psi \text{ and } Q \in \{\exists,\forall\} \end{cases}$$

Lemma $F.18$ shows that the rewritten formula $\phi^v_{s,u}$, for some $v \in \{\top,\bot\}$, is linear in the size of the original formula ϕ.

LEMMA F.17. *Let $M = \langle D,\Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be a partial M-state, $u \in U$ be a user, and ϕ be a D-formula. For all formulae ϕ and all $v \in \{\top,\bot\}$, $|\phi^v_{s,u}| \le (|extVoc(D, V)| + 1)\cdot|\phi|$.*

PROOF. Let $M = \langle D,\Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be a partial M-state, and $u \in U$ be a user. Let ϕ be an arbitrary formula over $D \cup V$ and v be an arbitrary value in $\{\top,\bot\}$. We now prove that $|\phi^v_{s,u}| \le m\cdot|\phi|$ by induction over the structure of the formula ϕ.

Base Case There are four cases:

1. $\phi := x = y$. In this case, $\phi^v_{s,u} = \phi$. From this, $|\phi^v_{s,u}| = |\phi|$. From this, it follows trivially that $|\phi^v_{s,u}| \le (|extVoc(D, V)| + 1)\cdot|\phi|$.
2. $\phi := \top$. The proof of this case is similar to that of $\phi := x = y$.
3. $\phi := \bot$. The proof of this case is similar to that of $\phi := x = y$.
4. $\phi := R(\overline{x})$. Without loss of generality, we assume that $v = \top$. From this, it follows that $\phi^\top_{s,u} := \bigvee_{S\in R^\top_{s,u}} S(\overline{x})$. From this, it follows that $|\phi^\top_{s,u}| = (|R^\top_{s,u}|-1)+\Sigma_{S\in R^\top_{s,u}} |S(\overline{x})|$. From this and $|S(\overline{x})| = 1 + |\overline{x}|$, it follows that $|\phi^\top_{s,u}| = (|R^\top_{s,u}| - 1) + \Sigma_{S\in R^\top_{s,u}}(1 + |\overline{x}|)$. From this, it follows that $|\phi^\top_{s,u}| = (|R^\top_{s,u}| - 1) + |R^\top_{s,u}|\cdot(1 + |\overline{x}|)$. From $\phi := R(\overline{x})$, it follows that $|\phi| = 1+|\overline{x}|$. From this and $|\phi^\top_{s,u}| = (|R^\top_{s,u}| - 1) + |R^\top_{s,u}|\cdot(1 + |\overline{x}|)$, it follows that $|\phi^\top_{s,u}| = |R^\top_{s,u}|\cdot|\phi| + (|R^\top_{s,u}| - 1)$. We claim that $|R^\top_{s,u}| \le |extVoc(D, V)|$. From this and $|\phi^\top_{s,u}| = |R^\top_{s,u}|\cdot |\phi|+(|R^\top_{s,u}|-1)$, it follows that $|\phi^\top_{s,u}| \le |extVoc(D, V)|\cdot |\phi| + |extVoc(D, V)|$. From this, it follows that $|\phi^\top_{s,u}| \le (|extVoc(D, V)| + 1)\cdot|\phi|$.
   We now prove our claim that $|R^\top_{s,u}| \le |extVoc(D, V)|$. The set $R^\top_{s,u}$ is a subset of $extVocabulary(D, V)$ by construction. The set $extVocabulary(D, V)$ contains any possible projection of tables in D and views in V. It is easy to check that the cardinality of $extVocabulary(D, V)$ is, indeed, $|extVoc(D, V)|$.

This completes the proof of the base case.

Induction Step Assume that our claim holds for all subformulae of ϕ. We now show that our claim holds also for ϕ. There are a number of cases depending on ϕ's structure.

1. $\phi := \psi\wedge\gamma$. From this, it follows that $\phi^v_{s,u} := \psi^v_{s,u}\wedge\gamma^v_{s,u}$. From this, it follows that $|\phi^v_{s,u}| = 1 + |\psi^v_{s,u}| + |\gamma^v_{s,u}|$. From the induction hypothesis, it follows that $|\psi^v_{s,u}| \le (|extVoc(D, V)|+1)\cdot|\psi|$ and $|\gamma^v_{s,u}| \le (|extVoc(D, V)| + 1)\cdot|\gamma|$. From this and $|\phi^v_{s,u}| = 1 + |\psi^v_{s,u}| + |\gamma^v_{s,u}|$, it follows that $|\phi^v_{s,u}| \le 1 + (|extVoc(D, V)| + 1)\cdot|\psi| + (|extVoc(D, V)|+1)\cdot|\gamma|$. From this and $|extVoc(D, V)| \ge 0$, it follows that $|\phi^v_{s,u}| \le |extVoc(D, V)|+1+(|extVoc(D, V)|+1)\cdot|\psi|+(|extVoc(D, V)|+1)\cdot|\gamma|$. From this, it follows that $|\phi^v_{s,u}| \le (|extVoc(D, V)|+1)\cdot(1+|\psi|+|\gamma|)$. From this and $|\phi| = 1+|\psi|+|\gamma|$, it follows that $|\phi^v_{s,u}| \le (|extVoc(D, V)| + 1)\cdot|\phi|$.
2. $\phi := \psi \vee \gamma$. The proof of this case is similar to that of $\phi := \psi \wedge \gamma$.
3. $\phi := \neg\psi$. From this, it follows that $\phi^v_{s,u} := \neg\psi^{\neg v}_{s,u}$. From this, it follows that $|\phi^v_{s,u}| = 1 + |\psi^{\neg v}_{s,u}|$. From the induction hypothesis, it follows that $|\psi^{\neg v}_{s,u}| \le (|extVoc(D, V)| + 1)\cdot|\psi|$. From this and $|\phi^v_{s,u}| = 1 + |\psi^v_{s,u}|$, it follows that $|\phi^v_{s,u}| \le 1+(|extVoc(D, V)|+1)\cdot|\psi|$. From this and $|extVoc(D, V)| \ge 0$, it follows that $|\phi^v_{s,u}| \le |extVoc(D, V)|+1+(|extVoc(D, V)|+1)\cdot|\psi|$. From this, it follows that $|\phi^v_{s,u}| \le (|extVoc(D, V)| + 1)\cdot(1 + |\psi|)$. From this and $|\phi| = 1 + |\psi|$, it follows that $|\phi^v_{s,u}| \le (|extVoc(D, V)| + 1)\cdot|\phi|$.
4. $\phi := \exists x.\,\psi$. If $\phi^v_{s,u}$ is $\neg v$, then the claim holds trivially since $|\phi^v_{s,u}| = 1$. In the following, we assume that $\phi^v_{s,u} := \exists x.\,\psi^v_{s,u}$. From this, it follows that $|\phi^v_{s,u}| = 2 + |\psi^v_{s,u}|$. From the induction hypothesis, it follows that $|\psi^v_{s,u}| \le (|extVoc(D, V)| + 1)\cdot|\psi|$. From this and $|\phi^v_{s,u}| = 2 + |\psi^v_{s,u}|$, it follows that $|\phi^v_{s,u}| \le 2 + (|extVoc(D, V)|+1)\cdot|\psi|$. From this and $|extVoc(D, V)| \ge 0$, it follows that $|\phi^v_{s,u}| \le 2\cdot|extVoc(D, V)| + 2 + (|extVoc(D, V)| + 1)\cdot|\psi|$. From this, it follows that $|\phi^v_{s,u}| \le (|extVoc(D, V)| + 1)\cdot(2 + |\psi|)$. From this and $|\phi| = 2 + |\psi|$, it follows that $|\phi^v_{s,u}| \le (|extVoc(D, V)| + 1)\cdot|\phi|$.
5. $\phi := \forall x.\,\psi$. The proof of this case is similar to that of $\phi := \exists x.\,\psi$.

This completes the proof of the induction step.

This completes the proof of our claim. □

LEMMA F.18. *Let $M = \langle D,\Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be a partial M-state, $u \in U$ be a user, and ϕ be a D-formula. For all sentences ϕ and all $v \in \{\top,\bot\}$, $|\phi^v_{s,u}| \le (|extVoc(D, V)|+1)\cdot|\phi|$ and $|\neg\phi^\top_{s,u}\wedge\phi^\bot_{s,u}| \le 2(|extVoc(D, V)| + 1)\cdot|\phi|$.*

PROOF. Let $M = \langle D,\Gamma\rangle$ be a system configuration, $s = \langle db, U, sec, T, V\rangle$ be a partial M-state, $u \in U$ be a user, and ϕ be a D-formula. Furthermore, let ϕ be a sentence and v be a value in $\{\top,\bot\}$. The fact that $|\phi^v_{s,u}| \le (|extVoc(D, V)|+1)\cdot|\phi|$ follows trivially from Lemma $F.17$. Let ψ be the formula $\neg\phi^\top_{s,u} \wedge \phi^\bot_{s,u}$. The size of ψ is $2 + |\phi^\top_{s,u}| + |\phi^\bot_{s,u}|$. From this and Lemma $F.17$, it follows that $|\psi| \le 2+(|extVoc(D, V)|+1)\cdot|\phi| + (|extVoc(D, V)| + 1)\cdot|\phi|$. From this, it follows that $|\psi| \le 2(|extVoc(D, V)| + 1)\cdot|\phi|$. This completes the proof. $\square$

In the following, we study the data complexity of our PDP. Note that, given a PDP f, the *data complexity of* f is the data complexity of the following decision problem:

Definition F.4. Let $M = \langle D, \Gamma\rangle$ be some fixed system configuration, $a \in \mathcal{A}_{D,U}$ be some fixed action, $u \in \mathcal{U}$ be some fixed user, $U \subseteq \mathcal{U}$ be some fixed set of users, $sec \in \Omega_{U,D}^{sec}$ be some fixed policy, T be some fixed set of triggers over D whose owners are in U, V be some fixed set of views over D whose owners are in U, and c be some fixed context.
INPUT: A database state db such that $\langle db, U, sec, T, V, c\rangle \in \Omega_M$.
Question: Is $f(\langle db, U, sec, T, V, c\rangle, a) = \top$? $\square$

We define in a similar way the data complexity of the *secure* procedure.

THEOREM F.2. *The data complexity of f_{conf}^{u} is AC^0.*

PROOF. Let $M = \langle D, \Gamma\rangle$ be some fixed system configuration, $a \in \mathcal{A}_{D,U}$ be some fixed action, $u \in \mathcal{U}$ be some fixed user, $U \subseteq \mathcal{U}$ be some fixed set of users, $sec \in \Omega_{U,D}^{sec}$ be some fixed policy, T be some fixed set of triggers over D whose owners are in U, V be some fixed set of views over D whose owners are in U, and c be some fixed context. The data complexity of f_{conf}^{u} is the maximum of the data complexities of $f_{conf,\mathtt{I},\mathtt{D}}^{u}$, $f_{conf,\mathtt{G}}^{u}$, and $f_{conf,\mathtt{S}}^{u}$. We claim that:
1. the data complexity of $f_{conf,\mathtt{I},\mathtt{D}}^{u}$ is AC^0,
2. the data complexity of $f_{conf,\mathtt{S}}^{u}$ is AC^0, and
3. the data complexity of $f_{conf,\mathtt{G}}^{u}$ is $O(1)$.

From this, it follows that the data complexity of f_{conf}^{u} is $max(AC^0, O(1))$. From this, it follows that the data complexity of f_{conf}^{u} is AC^0.

Our claims on the data complexity of $f_{conf,\mathtt{I},\mathtt{D}}^{u}$, $f_{conf,\mathtt{S}}^{u}$, and $f_{conf,\mathtt{G}}^{u}$ are proved respectively in Lemma *F*.19, Lemma *F*.21, and Lemma *F*.20. $\square$

LEMMA F.19. *The data complexity of $f_{conf,I,D}^{u}$ is AC^0.*

PROOF. Let $M = \langle D, \Gamma\rangle$ be some fixed system configuration, $a \in \mathcal{A}_{D,U}$ be some fixed `INSERT` or `DELETE` action, $u \in \mathcal{U}$ be some fixed user, $U \subseteq \mathcal{U}$ be some fixed set of users, $sec \in \Omega_{U,D}^{sec}$ be some fixed policy, T be some fixed set of triggers over D whose owners are in U, V be some fixed set of views over D whose owners are in U, and c be some fixed context. Furthermore, let $db \in \Omega_D^{\Gamma}$ be a database state such that $\langle db, U, sec, T, V, c\rangle \in \Omega_M$. We can check whether $f_{conf,\mathtt{I},\mathtt{D}}^{u}(\langle db, U, sec, T, V, c\rangle, a) = \top$ as follows:
1. If $trigger(s) = \epsilon$ and $a \notin \mathcal{A}_{D,u}$, return $\top$.
2. If $trigger(s) \neq \epsilon$ and $invoker(s) \neq u$, return $\top$.
3. Compute the result of $noLeak(s, a, u)$. If $noLeak(s, a, u) = \bot$, then returns $\bot$.
4. Compute the set $Dep(\Gamma, a)$.
5. Compute $secure(u, getInfo(a), s)$. If its result is $\bot$, return $\bot$.
6. For each $\gamma \in Dep(\Gamma, a)$, compute $secure(u, getInfoV(a, \gamma), s)$. If its result is $\bot$, return $\bot$.
7. For each $\gamma \in Dep(\Gamma, a)$, compute $secure(u, getInfoS(a, \gamma), s)$. If its result is $\bot$, return $\bot$.
8. Return $\top$.

The data complexity of the steps 1 and 2 is $O(1)$. We claim that also the data complexity of the third step is $O(1)$. The complexity of the fourth step is $O(|\Gamma|)$. From the definition of *getInfo*, the resulting formula is constant in the size of the database. Furthermore, also constructing the formula can be done in constant time in the size of the database. From this and Lemma *F*.22, it follows that the data complexity of the fifth step is AC^0. For a similar reason, the data complexity of the sixth and seventh steps is also AC^0. Therefore, the overall data complexity of the $f_{conf,\mathtt{I},\mathtt{D}}^{u}$ procedure is AC^0.

We now prove our claim that the data complexity of the *noLeak* procedure is $O(1)$. An algorithm implementing the *noLeak* procedure is as follows:
1. for each view $v \in V$, for each grant $g \in sec$, if $g = \langle op, u, \langle \mathtt{SELECT}, v\rangle, u'\rangle$, then
   (a) compute the set $tDet(v, s, M)$.
   (b) if $R \in tDet(v, s, M)$, for each $o \in tDet(v, s, M)$, check whether $\langle op, u, \langle \mathtt{SELECT}, o\rangle, u''\rangle \in sec$.

The size of the set $tDet(v, s, M)$ is at most $|D|$. From this, it follows that the complexity of the step 1.(b) is $O(|D| \cdot |sec|)$. From Lemma *E*.10 and the definition of *tDet*, the complexity of computing $tDet(v, s, M)$ is $O(|\phi|^3)$, where ϕ is v's definition. The overall complexity is, therefore, $O(|V| \cdot |sec| \cdot (|D| \cdot |sec| + 2^{|D|} \cdot |\phi|))$, where ϕ is the definition of the longest view in V. From this, it is easy to see that the data complexity of the *noLeak* procedure is $O(1)$. $\square$

LEMMA F.20. *The data complexity of $f_{conf,\mathtt{G}}^{u}$ is $O(1)$.*

PROOF. Let $M = \langle D, \Gamma\rangle$ be some fixed system configuration, $a \in \mathcal{A}_{D,U}$ be some fixed `GRANT` action, $u \in \mathcal{U}$ be some fixed user, $U \subseteq \mathcal{U}$ be some fixed set of users, $sec \in \Omega_{U,D}^{sec}$ be some fixed policy, T be some fixed set of triggers over D whose owners are in U, V be some fixed set of views over D whose owners are in U, and c be some fixed context. Furthermore, let $db \in \Omega_D^{\Gamma}$ be a database state such that $\langle db, U, sec, T, V, c\rangle \in \Omega_M$. We can check whether $f_{conf,\mathtt{G}}^{u}(\langle db, U, sec, T, V, c\rangle, \langle op, u'', p, u'\rangle)) = \top$ as follows.
1. If $trigger(s) = \epsilon$ and $a \notin \mathcal{A}_{D,u}$, return $\top$.
2. If $trigger(s) \neq \epsilon$ and $invoker(s) \neq u$, return $\top$.
3. If p is not a `SELECT` privilege, return $\top$.
4. If $u'' \neq u$, return $\top$.
5. For each $g \in sec$, if $g = \langle op, u, p, u'\rangle$, return $\top$.
6. Return $\bot$.

The complexity of the fifth step is $O(|sec|)$, whereas the complexity of the other steps is $O(1)$. Therefore, the overall complexity of the $f_{conf,\mathtt{G}}^{u}$ procedure is $O(|sec|)$. From this, it follows that the data complexity of $f_{conf,\mathtt{G}}^{u}$ procedure is $O(1)$. $\square$

LEMMA F.21. *The data complexity of $f_{conf,S}^{u}$ is AC^0.*

PROOF. Let $M = \langle D, \Gamma\rangle$ be some fixed system configuration, $a \in \mathcal{A}_{D,U}$ be some fixed `SELECT` action, $u \in \mathcal{U}$ be some fixed user, $U \subseteq \mathcal{U}$ be some fixed set of users, $sec \in \Omega_{U,D}^{sec}$ be some fixed policy, T be some fixed set of triggers over D whose owners are in U, V be some fixed set of views over D whose owners are in U, and c be some fixed context. Furthermore, let $db \in \Omega_D^{\Gamma}$ be a database state such that $\langle db, U, sec, T, V, c\rangle \in \Omega_M$. We can check whether $f_{conf,\mathtt{S}}^{u}(\langle db, U, sec, T, V, c\rangle, a)) = \top$ as follows.
1. If $trigger(s) = \epsilon$ and $a \notin \mathcal{A}_{D,u}$, return $\top$.
2. If $trigger(s) \neq \epsilon$ and $invoker(s) \neq u$, return $\top$.
3. Compute $secure(u, \phi, s)$ and return its result.

The complexity of the first and second steps is $O(1)$. From Lemma *F*.22, it follows that the data complexity of the third step is AC^0. From this, it follows that the data complexity of $f_{conf,\mathtt{S}}^{u}$ procedure is AC^0. $\square$

LEMMA F.22. *The data complexity of secure is AC^0.*

PROOF. Let $M = \langle D, \Gamma \rangle$ be some fixed system configuration, ϕ be some fixed sentence, $u \in \mathcal{U}$ be some fixed user, $U \subseteq \mathcal{U}$ be some fixed set of users, $sec \in \Omega^{sec}_{U,D}$ be some fixed policy, T be some fixed set of triggers over D whose owners are in U, V be some fixed set of views over D whose owners are in U, and c be some fixed context. Furthermore, let $db \in \Omega^{\Gamma}_{D}$ be a database state such that $\langle db, U, sec, T, V, c \rangle \in \Omega_M$. We denote by s the state $\langle db, U, sec, T, V, c \rangle$. We can check whether $secure(u, \phi, \langle db, U, sec, T, V, c \rangle) = \top$ as follows:

1. Compute the formula $\phi^{rw}_{s,u}$.
2. Compute $[\phi^{rw}_{s,u}]^{db}$.
3. $secure(u, \phi, \langle db, U, sec, T, V, c \rangle) = \top$ iff $[\phi^{rw}_{s,u}]^{db} = \bot$.

We claim that the first step can be done in constant time in terms of data complexity. It is well-known that the data complexity of query execution is AC^0 [3]. From this, it follows that the data complexity of *secure* is also AC^0.

We now prove our claim that computing the formula $\phi^{rw}_{s,u}$ can be done in constant time in terms of data complexity. The extended vocabulary $extVocabulary(D, V)$ does not depend on the database state. From this and the definition of R^v_s, where R is a predicate symbol and $v \in \{\top, \bot\}$, the set R^v_s (and the time needed to compute it) depends just on the database schema D and the set of views V. The set $AUTH_{s,u}$ and the time needed to compute it depend just on the size of the policy *sec*. Furthermore, the time needed to compute $AUTH^*_{s,u}$ depends just on the size of the policy *sec* and of the extended vocabulary. Therefore, for any predicate R, the set R^v_s can be computed in constant time in terms of database size. The computation of the formula ϕ', obtained by replacing sub-formulae of the form $\exists \overline{x}.R(\overline{x}, \overline{y})$ with the corresponding predicates in the extended vocabulary, can be done in linear time in terms of $|\phi|$ and in constant time in terms of $|db|$. Note that the size of the resulting formula is linear in $|\phi|$. It is easy to see that also computing $\phi^{\top}_{s,u}$ and $\phi^{\bot}_{s,u}$ can be done in linear time in terms of $|\phi|$ and in constant time in terms of $|db|$. As shown in Lemma $F.18$, the size of the resulting formula is linear in $|\phi|$. Finally, we can replace the predicates in the extended vocabulary with the corresponding sub-formulae again in linear time in terms of $|\phi|$. Note that, again, the size of the resulting formula is linear in $|\phi|$. Therefore, the overall rewriting process can be done in linear time in the size of ϕ and in constant time in the size of db. $\square$

G. COMPOSITION

Here, we model the PDP f, presented in Section 6, which is obtained by composing the PDPs f_{int} and f^u_{conf} presented above. The PDP f is obtained by composing f_{int} and f^u_{conf} as follows:

$$f(s, act) = f_{int}(s, act) \wedge f^{user(act,s)}_{conf}(s, act)$$

The function *user* takes as input an action and a state and returns the actual user executing the action. It is defined as follows, where i denotes the *invoker* function and tr denotes the *trigger* function.

$$user(act, s) = \begin{cases} i(s) & \text{if } tr(s) \neq \epsilon \\ u & \text{if } tr(s) = \epsilon \text{ and } act \in \mathcal{A}_{D,u} \end{cases}$$

We now show our main results, namely that (1) f provides both database integrity and data confidentiality, and (2) f's data complexity is AC^0.

THEOREM G.1. *Let M be a system configuration, f be as above, and $P = \langle M, f\rangle$ be an extended configuration.*

1. *For any user $u \in \mathcal{U}$, the PDP f provides data confidentiality with respect to $\vdash_u$, P, and u.*
2. *The PDP f provides database integrity with respect to P.*

PROOF. It follows from Lemma $G.1$ and Lemma $G.6$. □

THEOREM G.2. *The data complexity of f is AC^0.*

PROOF. From f's definition, it follows that f's data complexity is the maximum complexity between f^u_{conf}'s complexity and f_{int}'s complexity. From this, Theorem $E.2$, and Theorem $F.2$, it follows that the data complexity of f is AC^0. □

G.1 Database Integrity

Here, we show that f provides database integrity.

LEMMA G.1. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, f be as above, and $P = \langle M, f\rangle$ be an extended configuration. The PDP f provides database integrity with respect to P.*

PROOF. We prove the lemma by contradiction. Assume, for contradiction's sake, that f does not satisfy the database integrity property. There are three cases:

- there is a reachable state s and an action $act \in \mathcal{A}_{D,\mathcal{U}}$ such that $trigger(s) = \epsilon$, $f(s, act) = \top$, and $s \not\rightsquigarrow_{auth} act$. From $f(s, act) = \top$, it follows that $f_{int}(s, act) = \top$. From this fact, $trigger(s) = \epsilon$, and Lemma $E.5$, it follows $s \rightsquigarrow_{auth} act$, which leads to a contradiction.
- there is a reachable state s and a trigger $t \in \mathcal{TRIGGER}_D$ such that $trigger(s) = t$, $f(s, c) = \top$, $[\psi]^{s.db} = \bot$, and $s \not\rightsquigarrow_{auth} t$, where $c = \langle u, \texttt{SELECT}, \psi\rangle$ is t's condition. From $f(s, c) = \top$, it follows that $f_{int}(s, c) = \top$. From $f_{int}(s, c) = \top$, $[\psi]^{s.db} = \bot$, $trigger(s) = t$, and Lemma $E.7$, it follows $s \rightsquigarrow_{auth} t$, which leads to a contradiction.
- there is a reachable state s and a trigger $t \in \mathcal{TRIGGER}_D$ such that $trigger(s) = t$, $f(s, c) = \top$, $[\psi]^{s.db} = \top$, $f(s', a) = \top$, and $s \not\rightsquigarrow_{auth} t$, where $c = \langle u, \texttt{SELECT}, \psi\rangle$ is t's condition, a is t's action, and s' is the state obtained from s by updating the context's history. From $f(s', a) = \top$, it follows that $f_{int}(s', a) = \top$. Since s and s' are equivalent modulo the context's history and f_{int} does not depend on the context's history, it follows that $f_{int}(s, a) = \top$. From $f_{int}(s, c) = \top$, $[\psi]^{s.db} = \top$, $f_{int}(s, a) = \top$, $trigger(s) = t$, and Lemma $E.7$, it follows $s \rightsquigarrow_{auth} t$, which leads to a contradiction.

This completes the proof. □

LEMMA G.2. *Let $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is as above, and L be the P-LTS. For each reachable state $s = \langle db, U, sec, T, V, c\rangle$, $s \rightsquigarrow_{auth} g$ for all $g \in sec$.*

PROOF. The proof is very similar to that of Lemma $E.9$. □

G.2 Data Confidentiality

Here, we show that f provides the desired data confidentiality guarantees. First, we show that the PDP f', defined as $f'(s, act) := f^{user(act,s)}_{conf}(s, act)$, provides data confidentiality. Afterwards, we analyse the security of f.

In Lemma $G.3$ and Lemma $G.4$, we prove some preliminary results about f'. These results will then be used to prove f's security.

LEMMA G.3. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, a be an action in $\mathcal{A}_{D,\mathcal{U}}$, and $s, s' \in \Omega_M$ be two M-states such that $pState(s) \cong^{data}_{user(a,s),M} pState(s')$, $invoker(s) = invoker(s')$, and $trigger(s) = trigger(s')$. Then, $f^{user(a,s)}_{conf}(s, a) = \top$ iff $f^{user(a,s')}_{conf}(s', a) = \top$.*

PROOF. Let $s = \langle db, U, sec, T, V, c\rangle$ and $s' = \langle db', U', sec', T', V', c'\rangle$ be two M-states such that $pState(s) \cong^{data}_{M,user(a,s)} pState(s')$, $invoker(s) = invoker(s')$, and $trigger(s) = trigger(s')$.

We first show that $user(a, s) = user(a, s')$. Since $trigger(s) = trigger(s')$, there are two cases:

- $trigger(s) = \epsilon$. In this case, the result of $user(a, s)$ depends just on a. Therefore, $user(a, s) = user(a, s')$.
- $trigger(s) \neq \epsilon$. In this case, $user(a, s) = invoker(s)$ and $user(a, s') = invoker(s')$. From $invoker(s) = invoker(s')$, it follows that $user(a, s) = user(a, s')$.

Let u be the user $user(a, s)$. From Lemma $F.9$, it follows that $f^u_{conf}(s, a) = f^u_{conf}(s', a)$. This completes the proof. □

LEMMA G.4. *Let M be a system configuration, f' be as above, and $P = \langle M, f'\rangle$ be an extended configuration. For any user $u \in \mathcal{U}$, the PDP f' satisfies the data confidentiality property with respect to P, u, $\mathcal{ATK}_u$, and $\cong_{P,u}$.*

PROOF. It is easy to see that Lemmas F.9, F.11, F.12, F.13, F.14, F.15, and F.16 hold as well for f'. Therefore, we can easily adapt the proof of Theorem $F.1$ to f'. □

In Lemma $G.5$, we show that the PDP f returns the same result in any two data-indistinguishable states.

LEMMA G.5. *Let $M = \langle D, \Gamma\rangle$ be a system configuration, $s, s' \in \Omega_M$ be two M-states such that $pState(s) \cong^{data}_{M,user(a,s)} pState(s')$, $tuple(s) = tuple(s')$, $invoker(s) = invoker(s')$, and $trigger(s) = trigger(s')$, and f be the PDP as above. The following conditions hold:*

1. *If $trigger(s) = \epsilon$, for any action a in $\mathcal{A}_{D,\mathcal{U}}$, $f(s, a) = \top$ iff $f(s', a) = \top$.*
2. *If $trigger(s) \in \mathcal{TRIGGER}_D$, $f(s, trigCond(s)) = \top$ iff $f(s', trigCond(s)) = \top$.*
3. *If $trigger(s) \in \mathcal{TRIGGER}_D$, $trigCond(s) = \langle u, \texttt{SELECT}, \psi\rangle$, $[\psi]^{s.db} = [\psi]^{s'.db} = \top$, $f(s, trigAct(s)) = \top$ iff $f(s', trigAct(s')) = \top$.*

PROOF. We prove our three claims by contradiction.

1. Assume, for contradiction's sake, that there are two states s and s' and an action a such that $trigger(s) = trigger(s') = \epsilon$, $pState(s) \cong^{data}_{user(a,s),M} pState(s')$, $f(s,a) = \top$, and $f(s',a) = \bot$. From f's definition, $f(s,a) = \top$, $f(s',a) = \bot$, and Lemma $G.3$, it follows that $f_{int}(s,a) = \top$, $f_{int}(s',a) = \bot$, and $f^{user(a,s)}_{conf}(s,a) = f^{user(a,s')}_{conf}(s', a) = \top$. From this, it follows that $s' \not\rightsquigarrow^{approx}_{auth} a$. From $f_{int}(s,a) = \top$, it follows $s \rightsquigarrow^{approx}_{auth} a$. From this, $a \in \mathcal{A}_{D,\mathcal{U}}$, and Lemma $E.4$, it follows $s' \rightsquigarrow^{approx}_{auth} a$, which contradicts $s' \not\rightsquigarrow^{approx}_{auth} a$. This completes the proof for the first claim.
2. Assume, for contradiction's sake, that there are two states s and s' such that $trigger(s) = trigger(s')$, $trigger(s) \neq \epsilon$, $pState(s) \cong^{data}_{user(a,s),M} pState(s')$, $f(s,a) = \top$, and $f(s',a) = \bot$, where $trigCond(s) = trigCond(s') = a$. From f's definition, $f(s,a) = \top$, $f(s',a) = \bot$, and Lemma $G.3$, it follows that $f_{int}(s,a) = \top$, $f_{int}(s',a) = \bot$, and $f^{user(a,s)}_{conf}(s,a) = f^{user(a,s')}_{conf}(s',a) = \top$. From f_{int}'s definition, $trigger(s') \neq \epsilon$, and $a = trigCond(s')$, it follows that $f_{int}(s',a) = \top$, which contradicts $f_{int}(s', a) = \bot$. This completes the proof for the second claim.
3. Assume, for contradiction's sake, that there are two states s and s' such that $trigger(s) = trigger(s') = t$, $trigger(s) \neq \epsilon$, $pState(s) \cong^{data}_{user(a,s),M} pState(s')$, $[\psi]^{s.db} = [\psi]^{s'.db} = \top$, $f(s,a) = \top$, and $f(s',a) = \bot$, where $a = trigAct(s) = trigAct(s')$. From f's definition, $f(s,a) = \top$, $f(s',a) = \bot$, and Lemma $G.3$, it follows that $f^{user(a,s)}_{conf}(s,a) = f^{user(a,s')}_{conf}(s',a) = \top$, $f_{int}(s,a) = \top$, and $f_{int}(s',a) = \bot$. From this, it follows that $s' \not\rightsquigarrow^{approx}_{auth} t$. From $f_{int}(s,a) = \top$, it follows $s \rightsquigarrow^{approx}_{auth} t$. There are two cases depending on t's security mode:
   (a) $mode(t) = A$. From this and $s \rightsquigarrow^{approx}_{auth} t$, it follows that $s \rightsquigarrow^{approx}_{auth} a$ and $s \rightsquigarrow^{approx}_{auth} a'$, where $a' = getAction(statement(t), owner(t), tuple(s))$ is the trigger's action associated with the trigger's owner. Note that s and s' are data indistinguishable. From this, $a, a' \in \mathcal{A}_{D,\mathcal{U}}$, and Lemma $E.4$, it follows that $s' \rightsquigarrow^{approx}_{auth} a$ and $s' \rightsquigarrow^{approx}_{auth} a'$. From $s' \rightsquigarrow^{approx}_{auth} a$, $s' \rightsquigarrow^{approx}_{auth} a'$, $[\psi]^{s'.db} = \top$, and the rule `EXECUTE TRIGGER - 2`, it follows that $s' \rightsquigarrow^{approx}_{auth} t$, which contradicts $s' \not\rightsquigarrow^{approx}_{auth} t$.
   (b) $mode(t) = O$. From this and $s \rightsquigarrow^{approx}_{auth} t$, it follows that $s \rightsquigarrow^{approx}_{auth} a$. Note that s and s' are data indistinguishable. From this, $a, a' \in \mathcal{A}_{D,\mathcal{U}}$, and Lemma $E.4$, it follows that $s' \rightsquigarrow^{approx}_{auth} a$. From this, $[\psi]^{s'.db} = \top$, and the rule `EXECUTE TRIGGER - 1`, it follows that $s' \rightsquigarrow^{approx}_{auth} t$, which contradicts $s' \not\rightsquigarrow^{approx}_{auth} t$.

   This completes the proof for the third claim.

This completes the proof. $\square$

In Lemma $G.6$, we prove the main result of this section, namely that f provides data confidentiality. We first recall the concept of *derivation*. Given a judgment $r, i \vdash_u \phi$, a *derivation of* $r, i \vdash_u \phi$ *with respect to* $\mathcal{ATK}_u$, or *a derivation of* $r, i \vdash_u \phi$ for short, is a proof tree, obtained by applying the rules defining $\mathcal{ATK}_u$, that ends in $r, i \vdash_u \phi$. With a slight abuse of notation, we use $r, i \vdash_u \phi$ to denote both the judgment and its derivation. The length of a derivation, denoted $|r, i \vdash_u \phi|$, is the number of rule applications in it.

LEMMA G.6. *Let M be a system configuration, f be as above, and $P = \langle M, f\rangle$ be an extended configuration. For any user $u \in \mathcal{U}$, the PDP f provides data confidentiality with respect to P, u, $\mathcal{ATK}_u$, and $\cong_{P,u}$.*

PROOF. Let u be a user in $\mathcal{U}$, $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is as above, and L be the P-LTS. Furthermore, let r be a run in $traces(L)$, i be an integer such that $1 \leq i \leq |r|$, and ϕ be a sentence such that $r, i \vdash_u \phi$ holds. We claim that also $secure_{P,u}(r, i \vdash_u \phi)$ holds. The theorem follows trivially from the claim.

We now show that for all $r \in traces(L)$, all i such that $1 \leq i \leq |r|$, and all sentences ϕ such that $r, i \vdash_u \phi$ holds, then also $secure_{P,u}(r, i \vdash_u \phi)$ holds. We prove our claim by induction on the length of the derivation $r, i \vdash_u \phi$. In the following, we denote by e the function *extend*.

Base Case: Assume that $|r, i \vdash_u \phi| = 1$. There are a number of cases depending on the rule used to obtain $r, i \vdash_u \phi$.

1. *SELECT Success - 1.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{SELECT}, \phi\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = s'$, where $s' = \langle db, U, sec, T, V, c'\rangle$. From the rules, it follows that $f(s', \langle u, \texttt{SELECT}, \phi\rangle) = \top$. From this and f's definition, it follows that $f_{int}(s', \langle u, \texttt{SELECT}, \phi\rangle) = \top$ and $f^u_{conf}(s', \langle u, \texttt{SELECT}, \phi\rangle) = \top$, because $user(s', \langle u, \texttt{SELECT}, \phi\rangle) = u$. From $f^u_{conf}(s', \langle u, \texttt{SELECT}, \phi\rangle) = \top$, it follows $secure(u, \phi, s') = \top$. From this, Lemma $F.8$, and $pState(s) = pState(s')$, it follows $secure(u, \phi, s) = \top$. From this, Lemma $F.7$, and $last(r^i) = s$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.
2. *SELECT Success - 2.* The proof for this case is similar to that of *SELECT Success - 1*.
3. *INSERT Success.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $R(\overline{t})$. Then, $secure_{P,u}(r, i \vdash_u R(\overline{t}))$ holds. Indeed, in all runs r' (P, u)-indistinguishable from r^i the last action is $\langle u, \texttt{INSERT}, R, \overline{t}\rangle$. Furthermore, the action has been executed successfully. Therefore, according to the LTS rules, $\overline{t} \in last(r').db(R)$ for all runs $r' \in [\![r^i]\!]_{P,u}$. From this and the relational calculus semantics, it follows that $[R(\overline{t})]^{last(r').db} = \top$ for all runs $r' \in [\![r^i]\!]_{P,u}$. Hence, $secure_{P,u}(r, i \vdash_u R(\overline{t}))$ holds.
4. *INSERT Success - FD.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $\neg\exists \overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f(s', \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle) = \top$. From this and f's definition, it follows that $f^u_{conf}(s', \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle) = \top$, because $user(s', \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle) = u$. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$ holds because ϕ is equivalent to $getInfoS(\gamma, a)$ for some $\gamma \in Dep(\Gamma, a)$, where $a = \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i - 1 \vdash_u \phi)$ holds. We claim that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds. From this and Lemma $F.2$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.

   We now prove our claim that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds. Let s' be the state $last(r^{i-1})$. Furthermore, for brevity's sake, in the following we omit the *pState* function where needed. For instance, with a slight abuse of notation, we write $[\![s']\!]^{data}_{u,M}$ instead of $[\![pState(s')]\!]^{data}_{u,M}$. There are two cases:

(a) the `INSERT` command has caused an integrity constraint violation, i.e., $Ex(s) \neq \emptyset$. From $secure(u, \phi, s') = \top$ and Lemma $F.7$, it follows that $secure^{data}_{P,u}(r, i-1 \vdash_u \phi)$ holds. From this, it follows that $[\phi]^v = [\phi]^{s'}$ for any $v \in [\![s']\!]^{data}_{u,M}$. From this and the fact that the `INSERT` command caused an exception (i.e., $s' = s$), it follows that $[\phi]^v = [\phi]^s$ for any $v \in [\![s]\!]^{data}_{u,M}$. From this, it follows that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds.

(b) the `INSERT` command has not caused exceptions, i.e., $Ex(s) = \emptyset$. From $secure(u, \phi, s') = \top$ and Lemma $F.7$, it follows that $secure^{data}_{P,u}(r, i-1 \vdash_u \phi)$ holds. From this, it follows that $[\phi]^v = [\phi]^{s'}$ for any $v \in [\![s']\!]^{data}_{u,M}$. Furthermore, from F.7 and $Ex(s) = \emptyset$, it follows that ϕ holds in s'. Let $A_{s',R,\overline{t}}$ be the set $\{\langle db[R \oplus \overline{t}], U, sec, T, V\rangle \in \Pi_M \mid \exists db' \in \Omega_D. \langle db', U, sec, T, V\rangle \in [\![s']\!]^{data}_{M,u}\}$. It is easy to see that $[\![s]\!]^{data}_{M,u} \subseteq A_{s',R,\overline{t}}$. We now show that ϕ holds for any $z \in A_{s',R,\overline{t}}$. Let $z_1 \in [\![s']\!]^{data}_{M,u}$. From $[\phi]^v = [\phi]^{s'}$ for any $v \in [\![s']\!]^{data}_{u,M}$ and the fact that ϕ holds in s', it follows that $[\phi]^{z_1} = \top$. Therefore, for any $(\overline{k}_1, \overline{k}_2, \overline{k}_3) \in R(z_1)$ such that $|\overline{k}_1| = |\overline{v}|$, $|\overline{k}_2| = |\overline{w}|$, and $|\overline{k}_3| = |\overline{z}|$, if $k_1 = \overline{v}$, then $k_2 = \overline{w}$. Then, for any $(\overline{k}_1, \overline{k}_2, \overline{k}_3) \in R(z_1) \cup \{(\overline{v}, \overline{w}, \overline{q})\}$ such that $|\overline{k}_1| = |\overline{v}|$, $|\overline{k}_2| = |\overline{w}|$, and $|\overline{k}_3| = |\overline{z}|$, if $k_1 = \overline{v}$, then $k_2 = \overline{w}$. Therefore, ϕ holds also in $z_1[R \oplus \overline{t}] \in A_{pState(s'),R,\overline{t}}$. Hence, $[\phi]^z = \top$ for any $z \in A_{s',R,\overline{t}}$. From this and $[\![s]\!]^{data}_{M,u} \subseteq A_{s',R,\overline{t}}$, it follows that $[\phi]^z = \top$ for any $z \in [\![s]\!]^{data}_{M,u}$. From this, it follows that $secure^{data}_{P,u}(r, i, u, \phi)$ holds.

5. ***INSERT** Success - ID.* The proof of this case is similar to that for the ***INSERT** Success - FD.*
6. ***DELETE** Success.* The proof for this case is similar to that of ***INSERT** Success.*
7. ***DELETE** Success - ID.* The proof of this case is similar to that for the ***INSERT** Success - FD.*
8. ***INSERT** Exception.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSER}, R, \overline{t}\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $\neg R(\overline{t})$. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f(s', \langle u, \texttt{INSERT}, R, \overline{t}\rangle) = \top$. From this and f's definition, it follows that $f^u_{conf}(s', \langle u, \texttt{INSERT}, R, \overline{t}\rangle) = \top$, because $user(s', \langle u, \texttt{INSERT}, R, \overline{t}\rangle) = u$. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$ holds because $\phi = getInfo(\langle u, \texttt{INSERT}, R, \overline{t}\rangle)$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From the LTS semantics, it follows that $pState(s) \cong^{data}_{u,M} pState(last(r^{i-1}))$. From this, $secure(u, \phi, last(r^{i-1})) = \top$, and Lemma $F.8$, it follows that $secure(u, \phi, last(r^i)) = \top$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.
9. ***DELETE** Exception.* The proof for this case is similar to that of ***INSERT** Exception.*
10. ***INSERT** FD Exception.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U, sec, T, V, c'\rangle$, and ϕ be $\exists \overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f(s', \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle) = \top$. From this and f's definition, it follows that $f^u_{conf}(s', \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle) = \top$, because $user(s', \langle u, \texttt{INSERT}, R, \overline{t}\rangle) = u$. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$ because $\phi = getInfoV(\gamma, \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle)$ for some constraint $\gamma \in Dep(\Gamma, \langle u, \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle)$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From the LTS semantics, it follows that $pState(s) \cong^{data}_{u,M} pState(last(r^{i-1}))$. From this, Lemma $F.8$, and $secure(u, \phi, last(r^{i-1})) = \top$, it follows that $secure(u, \phi, last(r^i)) = \top$. From this and Lemma $F.7$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
11. ***INSERT** ID Exception.* The proof for this case is similar to that of ***INSERT** FD Exception.*
12. ***DELETE** FD Exception.* The proof for this case is similar to that of ***INSERT** FD Exception.*
13. *Integrity Constraint.* The proof of this case follows trivially from the fact that for any state $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and any $\gamma \in \Gamma$, $[\gamma]^{db} = \top$ by definition.
14. *Learn **GRANT/REVOKE** Backward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec', T, V, c'\rangle$, and t be a trigger whose `WHEN` condition is ϕ and whose action is either a `GRANT` or a `REVOKE`. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$, where u' is either the trigger's owner or the trigger's invoker depending on the security mode. From this and f's definition, it follows $f^u_{conf}(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$, because $user(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = u$ because t's invoker is u according to the rules. From this and f^u_{conf}'s definition, it follows $secure(u, \phi, last(r^{i-1})) = \top$. From this and F.7, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.
15. *Trigger **GRANT** Disabled Backward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec', T, V, c'\rangle$, and t be a trigger whose `WHEN` condition is ψ, and ϕ be $\neg\psi$. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$, where u' is either the trigger's owner or the trigger's invoker depending on the security mode. From this and f's definition, it follows $f^u_{conf}(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$, as $user(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = u$ because t's invoker is u according to the rules. From this and f^u_{conf}'s definition, it follows that also $secure(u, \phi, last(r^{i-1})) = \top$. From this and F.7, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.
16. *Trigger **REVOKE** Disabled Backward.* The proof for this case is similar to that of *Trigger **GRANT** Disabled Backward.*
17. *Trigger **INSERT** FD Exception.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec', T, V, c'\rangle$, and t be a trigger whose `WHEN` condition is ϕ and whose action act is a `INSERT` statement $\langle u', \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle$. Furthermore, let ϕ be $\exists \overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. From the rule's definition, it follows that $secEx(s) = \bot$. From this and the LTS rules, it follows that $f(last(r^{i-1}), act) = \top$. From this and f's definition, it follows that $f^u_{conf}(last(r^{i-1}), act) =$

$\top$, because $user(last(r^{i-1}), act) = u$ because t's invoker is u according to the rules. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$ because $\phi = getInfoV(\gamma, act)$ for some constraint $\gamma \in Dep(\Gamma, act)$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.

18. *Trigger* `INSERT` *ID Exception.* The proof for this case is similar to that of *Trigger* `INSERT` *ID Exception.*
19. *Trigger* `DELETE` *ID Exception.* The proof for this case is similar to that of *Trigger* `DELETE` *ID Exception.*
20. *Trigger Exception.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec', T, V, c'\rangle$, and t be a trigger whose `WHEN` condition is ϕ and whose action is act. From the rule's definition, it follows that $f(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$, where u' is either the trigger's owner or the trigger's invoker depending on the security mode. From this and f's definition, it follows $f^u_{conf}(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$, because $user(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = u$ since t's invoker is u according to the rules. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$. From this and F.7, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.
21. *Trigger* `INSERT` *Exception.* The proof for this case is similar to that of `INSERT` *Exception.*
22. *Trigger* `DELETE` *Exception.* The proof for this case is similar to that of `DELETE` *Exception.*
23. *Trigger Rollback* `INSERT`. Let i be such that $r^i = r^{i-n-1} \cdot \langle u, \texttt{INSERT}, R, \bar{t}\rangle \cdot s_1 \cdot t_1 \cdot s_2 \cdot \ldots \cdot t_n \cdot s_n$, where $s_1, s_2, \ldots, s_n \in \Omega_M$ and $t_1, \ldots, t_n \in \mathcal{TRIGGER}_D$, and ϕ be $\neg R(\bar{t})$. Furthermore, let $last(r^{i-n-1}) = \langle db', U', sec', T', V', c'\rangle$ and s_n be $\langle db, U, sec, T, V, c\rangle$. From the rule's definition, it follows that $secEx(s_1) = \bot$. From this, it follows that $f(last(r^{i-n-1}), \langle u, \texttt{INSERT}, R, \bar{t}\rangle) = \top$. From this and f's definition, it follows $f^u_{conf}(last(r^{i-n-1}), \langle u, \texttt{INSERT}, R, \bar{t}\rangle) = \top$ since $user(last(r^{i-n-1}), \langle u, \texttt{INSERT}, R, \bar{t}\rangle) = u$. From this and f^u_{conf}'s definition, it follows $secure(u, \phi, last(r^{i-n-1})) = \top$ because $\phi = getInfo(\langle u, \texttt{INSERT}, R, \bar{t}\rangle)$. From the LTS semantics, it follows that $last(r^{i-n-1}) \cong^{data}_{M,u} s_n$ because $pState(last(r^{i-n-1})) = pState(s_n)$. From this, Lemma $F.8$, and $secure(u, \phi, last(r^{i-n-1})) = \top$, it follows $secure(u, \phi, s_n) = \top$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.
24. *Trigger Rollback* `DELETE`. The proof for this case is similar to that of *Trigger Rollback* `INSERT`.

This completes the proof of the base step.

Induction Step: Assume that the claim hold for any derivation of $r, j \vdash_u \psi$ such that $|r, j \vdash_u \psi| < |r, i \vdash_u \phi|$. We now prove that the claim also holds for $r, i \vdash_u \phi$. There are a number of cases depending on the rule used to obtain $r, i \vdash_u \phi$.

1. *View.* The proof of this case follows trivially from the semantics of the relational calculus extended over views.
2. *Propagate Forward* `SELECT`. Let i be such that $r^{i+1} = r^i \cdot \langle u, \texttt{SELECT}, \psi\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^i) = \langle db', U', sec', T', V', c'\rangle$. From the rule, it follows that $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $G.7$, the action $\langle u, \texttt{SELECT}, \psi\rangle$ preserves the equivalence class with respect to r^i, P, and u. From this, Lemma $F.12$, and $secure_{P,u}(r, i \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i+1 \vdash_u \phi)$ holds.
3. *Propagate Forward* `GRANT/REVOKE`. Let i be such that $r^{i+1} = r^i \cdot \langle op, u', p, u\rangle \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^i) = \langle db', U', sec', T', V', c'\rangle$. From the rule, it follows that $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $G.7$, the action $\langle op, u', p, u\rangle$ preserves the equivalence class with respect to r^i, P, and u. From this, Lemma $F.13$, and $secure_{P,u}(r, i \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i+1 \vdash_u \phi)$ holds.
4. *Propagate Forward* `CREATE`. The proof for this case is similar to that of *Propagate Forward* `SELECT`.
5. *Propagate Backward* `SELECT`. Let i be such that $r^{i+1} = r^i \cdot \langle u, \texttt{SELECT}, \psi\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$ and $last(r^i) = \langle db, U, sec, T, V, c\rangle$. From the rule, it follows that $r, i+1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i+1 \vdash_u \phi)$ holds. From Lemma $G.7$, the action $\langle u, \texttt{SELECT}, \psi\rangle$ preserves the equivalence class with respect to r^i, P, and u. From this, Lemma $F.12$, and $secure_{P,u}(r, i+1 \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
6. *Propagate Backward* `GRANT/REVOKE`. Let i be such that $r^{i+1} = r^i \cdot \langle op, u', p, u\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$ and $last(r^i) = \langle db, U, sec, T, V, c\rangle$. From the rule, it follows that $r, i+1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i+1 \vdash_u \phi)$ holds. From Lemma $G.7$, the action $\langle op, u', p, u\rangle$ preserves the equivalence class with respect to r^i, P, and u. From this, Lemma $F.13$, and $secure_{P,u}(r, i+1 \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
7. *Propagate Backward* `CREATE TRIGGER`. The proof for this case is similar to that of *Propagate Backward* `SELECT`.
8. *Propagate Backward* `CREATE VIEW`. Note that the formulae ψ and $replace(\psi, o)$ are semantically equivalent. This is the only difference between the proof for this case and the one for the *Propagate Backward* `SELECT` case.
9. *Rollback Backward - 1.* Let i be such that $r^i = r^{i-n-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s_1 \cdot t_1 \cdot s_2 \cdot \ldots \cdot t_n \cdot s_n$, where $s_1, s_2, \ldots, s_n \in \Omega_M$, $t_1, \ldots, t_n \in \mathcal{TRIGGER}_D$, and op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$. Furthermore, let s_n be $\langle db', U', sec', T', V', c'\rangle$ and $last(r^{i-n-1})$ be $\langle db, U, sec, T, V, c\rangle$. From the rule's definition, $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $G.8$, the triggers t_j preserve the equivalence class with respect to $r^{i-n-1+j}$, P, and u for any $1 \leq j \leq n$. Therefore, for any $v \in [\![r^{i-1}]\!]_{P,u}$, the run $e(v, t_n)$ contains the roll-back. Therefore, for any $v \in [\![r^{i-1}]\!]_{P,u}$, the state $last(e(v, t_n))$ is the state just before the action $\langle u, op, R, \bar{t}\rangle$. Let A be the set of partial states associated with the roll-back states. It is easy to see that A is the same as $\{pState(last(t')) | t' \in [\![r^{i-n-1}]\!]_{P,u}\}$. From $secure_{P,u}(r, i \vdash_u \phi)$, it follows that ϕ has the same result over all states in A. From this and $A = \{pState(last(t')) | t' \in [\![r^{i-n-1}]\!]_{P,u}\}$, it follows that ϕ has the same result over all states in $\{pState(last(t')) | t' \in [\![r^{i-n-1}]\!]_{P,u}\}$. From this, it follows that $secure_{P,u}(r, i-n-1 \vdash_u \phi)$ holds.
10. *Rollback Backward - 2.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \bar{t}\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and op is one of

$\{\texttt{INSERT}, \texttt{DELETE}\}$. From the rule's definition, $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $G.7$, the action $\langle u, op, R, \overline{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. From this, Lemma $F.11$, the fact that the action does not modify the database state, and $secure_{P,u}(r, i \vdash_u \phi)$, it follows $secure_{P,u}(r, i-1 \vdash_u \phi)$.

11. *Rollback Forward - 1.* Let i be such that $r^i = r^{i-n-1} \cdot \langle u, op, R, \overline{t}\rangle \cdot s_1 \cdot t_1 \cdot s_2 \cdot \ldots \cdot t_n \cdot s_n$, where $s_1, s_2, \ldots, s_n \in \Omega_M$, $t_1, \ldots, t_n \in \mathcal{TRIGGER}_D$, and op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$. Furthermore, let s_n be $\langle db, U, sec, T, V, c\rangle$ and $last(r^{i-n-1})$ be $\langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-n-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-n-1 \vdash_u \phi)$ holds. From Lemma $G.8$, the triggers t_j preserve the equivalence class with respect to $r^{i-n-1+j}$, P, and u for any $1 \leq j \leq n$. Independently on the cause of the roll-back (either a security exception or an integrity constraint violation), we claim that the set A of roll-back partial states is $\{pState(last(t')) | t' \in [\![r^{i-n-1}]\!]_{P,u}\}$. From $secure_{P,u}(r, i-n-1 \vdash_u \phi)$, the result of ϕ is the same for all states in A. From this and $A = \{pState(last(t')) | t' \in [\![r^{i-n-1}]\!]_{P,u}\}$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
We now prove our claim. It is trivial to see (from the LTS's semantics) that the set of rollback's states is a subset of $\{last(v) | v \in [\![r^{i-n-1}]\!]_{P,u}\}$. Assume, for contradiction's sake, that there is a state in $\{last(v) | v \in [\![r^{i-n-1}]\!]_{P,u}\}$ that is not a rollback state for the runs in $[\![r^i]\!]_{P,u}$. This is impossible since all triggers $t_1, \ldots, t_n$ preserve the equivalence class.
12. *Rollback Forward - 2.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \overline{t}\rangle \cdot s$, where $op \in \{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From Lemma $G.7$, the action $\langle u, op, R, \overline{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. From this, Lemma $F.11$, the fact that the action does not modify the database state, and $secure_{P,u}(r, i-1 \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
13. *Propagate Forward* ***INSERT/DELETE*** *Success.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \overline{t}\rangle \cdot s$, where $op \in \{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From Lemma $G.7$, the action $\langle u, op, R, \overline{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. From $reviseBelif(r^{i-1}, \phi, r^i)$, it follows that the execution of $\langle u, op, R, \overline{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ for any $v \in [\![r^{i-1}]\!]_{P,u}$. From this, Lemma $F.11$, and $secure_{P,u}(r, i-1 \vdash_u \phi)$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.
14. *Propagate Forward* ***INSERT*** *Success - 1.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \overline{t}\rangle \cdot s$, where op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From Lemma $G.7$, the action $\langle u, op, R, \overline{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. We claim that the execution of $\langle u, \texttt{INSERT}, R, \overline{t}\rangle$ does not alter the content of the tables in $tables(\phi)$. From this, Lemma $F.11$, and $secure_{P,u}(r, i-1 \vdash_u \phi)$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.
We now prove our claim that the execution of $\langle u, \texttt{INSERT}, R, \overline{t}\rangle$ does not alter the content of the tables in $tables(\phi)$. From the rule's definition, it follows that $r, i-1 \vdash_u R(\overline{t})$ holds. From this and Lemma $B.1$, it follows that $[R(\overline{t})]^{last(r^{i-1}).db} = \top$. From $r, i-1 \vdash_u R(\overline{t})$ and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u R(\overline{t}))$ holds. From this and $[R(\overline{t})]^{last(r^{i-1}).db} = \top$, it follows that $[R(\overline{t})]^{last(v).db} = \top$ for any $v \in [\![r^{i-1}]\!]_{P,u}$. From this and the relational calculus semantics, it follows that the execution of $\langle u, op, R, \overline{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ for any $v \in [\![r^{i-1}]\!]_{P,u}$.
15. *Propagate Forward* ***DELETE*** *Success - 1.* The proof for this case is similar to that of *Propagate Forward* ***INSERT*** *Success - 1.*
16. *Propagate Backward* ***INSERT/DELETE*** *Success.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \overline{t}\rangle \cdot s$, where $op \in \{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $G.7$, the action $\langle u, op, R, \overline{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. From $reviseBelif(r^{i-1}, \phi, r^i)$, it follows that the execution of $\langle u, op, R, \overline{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ for any $v \in [\![r^{i-1}]\!]_{P,u}$. From this, Lemma $F.11$, and $secure_{P,u}(r, i \vdash_u \phi)$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.
17. *Propagate Backward* ***INSERT*** *Success - 1.* Let i be such that $r^i = r^{i-1} \cdot \langle u, op, R, \overline{t}\rangle \cdot s$, where op is one of $\{\texttt{INSERT}, \texttt{DELETE}\}$, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds. From Lemma $G.7$, the action $\langle u, op, R, \overline{t}\rangle$ preserves the equivalence class with respect to r^{i-1}, P, and u. We claim that the execution of $\langle u, \texttt{INSERT}, R, \overline{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ for any $v \in [\![r^{i-1}]\!]_{P,u}$ (the proof of this claim is in the proof of the *Propagate Forward* ***INSERT*** *Success - 1* case). From this, Lemma $F.11$, and $secure_{P,u}(r, i \vdash_u \phi)$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.
18. *Propagate Backward* ***DELETE*** *Success - 1.* The proof for this case is similar to that of *Propagate Forward* ***DELETE*** *Success - 1.*
19. *Reasoning.* Let Δ be a subset of $\{\delta \,|\, r, i \vdash_u \delta\}$ and $last(r^i) = \langle db, U, sec, T, V, c\rangle$. From the induction hypothesis, it follows that $secure_{P,u}(r, i \vdash_u \delta)$ holds for any $\delta \in \Delta$. Note that, given any $\delta \in \Delta$, from $r, i \vdash_u \delta$ and Lemma $B.1$, it follows that δ holds in $last(r^i)$. From this, $secure_{P,u}(r, i \vdash_u \delta)$ holds for any $\delta \in \Delta$, $\Delta \models_{fin} \phi$, and Lemma $F.10$, it follows that $secure_{P,u}(r, i \vdash_u \phi)$ holds.
20. *Learn* ***INSERT*** *Backward - 3.* Let i be such that $r^i = r^{i-1} \cdot \langle u, \texttt{INSERT}, R, \overline{t}\rangle \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and ϕ be $\neg R(\overline{t})$. From the rule's definition, $secEx(s) = \bot$. From this and the LTS rules, it follows that $f(last(r^{i-1}), \langle u,$

$\mathtt{INSERT}, R, \overline{t}\rangle) = \top$. From this and f's definition, it follows that $f^{u}_{conf}(last(r^{i-1}), \langle u, \mathtt{INSERT}, R, \overline{t}\rangle) = \top$ because $user(last(r^{i-1}), \langle u, \mathtt{INSERT}, R, \overline{t}\rangle) = u$. From this and f^{u}_{conf}'s definition, it follows $secure(u, \phi, last(r^{i-1})) = \top$ because $\phi = getInfo(\langle u, \mathtt{INSERT}, R, \overline{t}\rangle)$. From this and Lemma $F.7$, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.

21. *Learn* `DELETE` *Backward - 3.* The proof for this case is similar to that of *Learn* `INSERT` *Backward - 3.*
22. *Propagate Forward Disabled Trigger.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and t be a trigger. Furthermore, let ψ be t's condition where all free variables are replaced with $tpl(last(r^{i-1}))$. From the rule, it follows that $r, i-1 \vdash_u \phi$. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. Furthermore, from Lemma $G.8$, it follows that t preserves the equivalence class with respect to r^{i-1}, P, and u. If the trigger's action is an `INSERT` or a `DELETE` operation, we claim that the operation does not change the content of any table in $tables(\phi)$ for any run $v \in [\![r^{i-1}]\!]_{P,u}$. From this, the fact that t preserves the equivalence class with respect to r^{i-1}, P, and u, Lemma $F.14$, and $secure_{P,u}(r, i-1 \vdash_u \phi)$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
    We now prove our claim. Assume that t's action in either an `INSERT` or a `DELETE` operation. From the rule, it follows that $r, i-1 \vdash_u \neg\psi$. From this and Lemma $B.1$, $[\psi]^{last(r^{i-1})} = \bot$. From $r, i-1 \vdash_u \neg\psi$ and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \psi)$ holds. From this and $[\psi]^{last(r^{i-1}).db} = \bot$, it follows that $[\psi]^{v.db} = \bot$ for any run $v \in [\![r^{i-1}]\!]_{P,u}$. Therefore, the trigger t is disabled in any run $v \in [\![r^{i-1}]\!]_{P,u}$. From this and the LTS semantics, it follows that t's execution does not change the content of any table in $tables(\phi)$ for any run $v \in [\![r^{i-1}]\!]_{P,u}$.
23. *Propagate Backward Disabled Trigger.* The proof for this case is similar to that of *Propagate Forward Disabled Trigger.*
24. *Learn* `INSERT` *Forward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and t be a trigger, and ϕ be $R(\overline{t})$. Furthermore, let ψ be t's condition where all free variables are replaced with $tpl(last(r^{i-1}))$. From the rule's definition, it follows that t's action is $\langle u', \mathtt{INSERT}, R, \overline{t}\rangle$ and that $r, i-1 \vdash_u \psi$ holds. From Lemma $B.1$ and $r, i-1 \vdash_u \psi$, it follows that $[\psi]^{last(r^{i-1}).db} = \top$. From this, $secEx(s) = \bot$, and $Ex(s) = \emptyset$, it follows that t's action has been executed successfully. From this, it follows that $\overline{t} \in s.db(R)$. From $r, i-1 \vdash_u \psi$ and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \psi)$. From this and $[\psi]^{last(r^{i-1}).db} = \top$, it follows that $[\psi]^{last(v).db} = \top$ for any $v \in [\![r^{i-1}]\!]_{P,u}$. From this, it follows that the trigger t is enabled in any run $v \in [\![r^{i-1}]\!]_{P,u}$. From Lemma $G.8$, it follows that t preserves the equivalence class with respect to r^{i-1}, P, and u. From this, $secEx(s) = \bot$, $Ex(s) = \emptyset$, and the fact that the trigger t is enabled in any run $v \in [\![r^{i-1}]\!]_{P,u}$, it follows that t's action is executed successfully in any run $e(v, t)$, where $v \in [\![r^{i-1}]\!]_{P,u}$. From this, it follows that $db''(R)$, where $db'' = \overline{t} \in last(e(v, t)).db$, for any $v \in [\![r^{i-1}]\!]_{P,u}$. Therefore, $secure_{P,u}(r, i \vdash_u \phi)$ holds.
25. *Learn* `INSERT` *- FD.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$, and $t \in \mathcal{TRIGGER}_D$, and ϕ be $\neg\exists\overline{y}, \overline{z}.\, R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. Furthermore, let ψ be t's condition where all free variables are replaced with the values in $tpl(last(r^{i-1}))$ and $\langle u', \mathtt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle$ be t's actual action. From the rule, it follows that $r, i-1 \vdash_u \psi$. From this and Lemma $B.1$, it follows that $[\psi]^{last(r^{i-1}).db} = \top$. From this, $Ex(s) = \emptyset$, and $secEx(s) = \bot$, it follows that $f(s', \langle u', \mathtt{INSERT}, R, \overline{t}\rangle) = \top$, where s' is the state just after the execution of the `SELECT` statement associated with t's `WHEN` clause. From this and f's definition, it follows that $f^{u}_{conf}(s', \langle u', \mathtt{INSERT}, R, \overline{t}\rangle) = \top$ because $user(s', \langle u', \mathtt{INSERT}, R, \overline{t}\rangle) = u$ since u is t's invoker. From this and f^{u}_{conf}'s definition, it follows that $secure(u, \phi, s') = \top$. From this, $pState(s') = pState(last(r^{i-1}))$, and Lemma $F.8$, it follows $secure(u, \phi, last(r^{i-1})) = \top$. From this and Lemma $F.7$, it follows $secure_{P,u}(r, i-1 \vdash_u \phi)$. We claim that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds. From this and Lemma $F.2$, it follows that also $secure_{P,u}(r, i \vdash_u \phi)$ holds.
    We now prove our claim that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds. Let s' be the state just after the execution of the `SELECT` statement associated with t's `WHEN` clause and s'' be the state $last(r^{i-1})$. Furthermore, for brevity's sake, in the following we omit the $pState$ function where needed. For instance, with a slight abuse of notation, we write $[\![s']\!]^{data}_{u,M}$ instead of $[\![pState(s')]\!]^{data}_{u,M}$. From $secure(u, \phi, s') = \top$, $s' \cong^{data}_{M,u} s''$, Lemma $F.8$, and Lemma $F.7$, it follows that $secure^{data}_{P,u}(r, i-1 \vdash_u \phi)$ holds. From this, it follows that $[\phi]^v = [\phi]^{s''}$ for any $v \in [\![s'']\!]^{data}_{u,M}$. Furthermore, from Proposition $F.7$ and $Ex(s) = \emptyset$, it follows that ϕ holds in s''. Let $A_{s'',R,\overline{t}}$ be the set $\{\langle db[R \oplus \overline{t}], U, sec, T, V\rangle \in \Pi_M \mid \exists db' \in \Omega_D.\, \langle db', U, sec, T, V\rangle \in [\![s'']\!]^{data}_{M,u}\}$. It is easy to see that $[\![s]\!]^{data}_{M,u} \subseteq A_{s'',R,\overline{t}}$. We now show that ϕ holds for any $z \in A_{s'',R,\overline{t}}$. Let $z_1 \in [\![s'']\!]^{data}_{M,u}$. From $[\phi]^v = [\phi]^{s''}$ for any $v \in [\![s'']\!]^{data}_{u,M}$ and the fact that ϕ holds in s'', it follows that $[\phi]^{z_1} = \top$. Therefore, for any $(\overline{k}_1, \overline{k}_2, \overline{k}_3) \in R(z_1)$ such that $|\overline{k}_1| = |\overline{v}|$, $|\overline{k}_2| = |\overline{w}|$, and $|\overline{k}_3| = |\overline{q}|$, if $k_1 = \overline{v}$, then $k_2 = \overline{w}$. Then, for any $(\overline{k}_1, \overline{k}_2, \overline{k}_3) \in R(z_1) \cup \{(\overline{v}, \overline{w}, \overline{q})\}$ such that $|\overline{k}_1| = |\overline{v}|$, $|\overline{k}_2| = |\overline{w}|$, and $|\overline{k}_3| = |\overline{q}|$, if $k_1 = \overline{v}$, then $k_2 = \overline{w}$. Therefore, ϕ holds also in $z_1[R \oplus \overline{t}] \in A_{pState(s''),R,\overline{t}}$. Hence, $[\phi]^z = \top$ for any $z \in A_{s'',R,\overline{t}}$. From this and $[\![s]\!]^{data}_{M,u} \subseteq A_{s'',R,\overline{t}}$, it follows that $[\phi]^z = \top$ for any $z \in [\![s]\!]^{data}_{M,u}$. From this, it follows that $secure^{data}_{P,u}(r, i \vdash_u \phi)$ holds.
26. *Learn* `INSERT` *- FD - 1.* The proof of this case is similar to that of *Learn* `INSERT` *- FD.*
27. *Learn* `INSERT` *- ID.* The proof of this case is similar to that of *Learn* `INSERT` *- FD.* See also the proof of `INSERT` *Success - ID.*
28. *Learn* `INSERT` *- ID - 1.* The proof of this case is similar to that of *Learn* `INSERT` *- ID.*
29. *Learn* `INSERT` *Backward - 1.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and $t \in \mathcal{TRIGGER}_D$, and ϕ be t's actual `WHEN` condition, where all free variables are replaced with the values in $tpl(last(r^{i-1}))$. From the rule's definition, it follows that $secEx(s) = \top$. From this, the LTS semantics, and $secEx(s) = \top$, it fol-

lows that $f(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$. From this and f's definition, it follows $f^u_{conf}(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$ because $user(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = u$ since u is t's invoker. From this and f^u_{conf}'s definition, it follows that $secure(u, \phi, last(r^{i-1})) = \top$. From this and Lemma F.7, it follows that also $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds.

30. *Learn* **INSERT** *Backward - 2.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and $t \in \mathcal{TRIGGER}_D$, and ϕ be $\neg R(\overline{t})$. Furthermore, let $act = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$ be t's actual action and γ be t's actual WHEN condition obtained by replacing all free variables with the values in $tpl(last(r^{i-1}))$. From the rule's definition, it follows $secEx(s) = \top$ and there is a ψ such that $r, i-1 \vdash_u \psi$ and $r, i \vdash_u \neg\psi$. We claim that $[\gamma]^{db} = \top$. From this and $secEx(s) = \top$, it follows that $f(s', \langle u', \texttt{INSERT}, R, \overline{t}\rangle) = \top$, where s' is the state obtained after the evaluation of t's WHEN condition. From this and f's definition, it follows $f^u_{conf}(s', \langle u', \texttt{INSERT}, R, \overline{t}\rangle) = \top$ as $user(s', \langle u', \texttt{INSERT}, R, \overline{t}\rangle) = u$ because u is t's invoker. From this and f^u_{conf}'s definition, it follows $secure(u, \phi, s') = \top$ since ϕ is equivalent to $getInfo(\langle u', \texttt{INSERT}, R, \overline{t}\rangle)$. From this, Lemma F.8, and $pState(s') = pState(last(r^{i-1}))$, it follows $secure(u, \phi, last(r^{i-1})) = \top$. From this and Lemma F.7, it follows $secure_{P,u}(r, i-1 \vdash_u \phi)$.
    We now prove our claim that $[\gamma]^{db} = \top$. Assume, for contradiction's sake, that this is not the case. From this and the LTS rules, it follows that $db = db'$. From the rule's definition, it follows that there is a ψ such that $r, i-1 \vdash_u \psi$ and $r, i \vdash_u \neg\psi$. From this, Lemma B.1, $s = \langle db', U', sec', T', V', c'\rangle$, and $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, it follows that $[\psi]^{db} = \top$ and $[\neg\psi]^{db'} = \top$. Therefore, $[\psi]^{db} = \top$ and $[\psi]^{db'} = \bot$. Hence, $db \neq db'$, which contradicts $db = db'$.
31. *Learn* **DELETE** *Forward.* The proof of this case is similar to that of *Learn* **INSERT** *Forward.*
32. *Learn* **DELETE** *- ID.* The proof of this case is similar to that of *Learn* **INSERT** *- FD.* See also the proof of **DELETE** *Success - ID.*
33. *Learn* **DELETE** *- ID - 1.* The proof of this case is similar to that of *Learn* **DELETE** *- ID.*
34. *Learn* **DELETE** *Backward - 1.* The proof of this case is similar to that of *Learn* **INSERT** *Backward - 1.*
35. *Learn* **DELETE** *Backward - 2.* The proof of this case is similar to that of *Learn* **INSERT** *Backward - 2.*
36. *Propagate Forward Trigger Action.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where t is a trigger, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From Lemma G.8, the trigger t preserves the equivalence class with respect to r^{i-1}, P, and u. We claim that the execution of t does not alter the content of the tables in $tables(\phi)$. From this, Lemma F.11, and $secure_{P,u}(r, i-1 \vdash_u \phi)$, it follows $secure_{P,u}(r, i \vdash_u \phi)$.
    We now prove our claim that the execution of t does not alter the content of the tables in $tables(\phi)$. If the trigger is not enabled, the claim is trivial. In the following, we assume the trigger is enabled. There are four cases:
    - t's action is an INSERT statement. This case amount to claiming that the INSERT statement $\langle u', \texttt{INSERT}, R, \overline{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ in case $reviseBelif(r^{i-1}, \phi, r^i) = \top$. We proved the claim above in the *Propagate Forward* **INSERT/DELETE** *Success* case.
    - t's action is an DELETE statement. The proof is similar to that of the INSERT case.
    - t's action is an GRANT statement. In this case, the action does not alter the database state and the claim follows trivially.
    - t's action is an REVOKE statement. The proof is similar to that of the GRANT case.
37. *Propagate Backward Trigger Action.* The proof of this case is similar to *Propagate Backward Trigger Action.*
38. *Propagate Forward* **INSERT** *Trigger Action.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where t is a trigger, $s = \langle db, U, sec, T, V, c\rangle \in \Omega_M$ and $last(r^{i-1}) = \langle db', U', sec', T', V', c'\rangle$. From the rule's definition, $r, i-1 \vdash_u \phi$ holds. From this and the induction hypothesis, it follows that $secure_{P,u}(r, i-1 \vdash_u \phi)$ holds. From Lemma G.8, the trigger t preserves the equivalence class with respect to r^{i-1}, P, and u. We claim that the execution of t does not alter the content of the tables in $tables(\phi)$. From this, Lemma F.11, and $secure_{P,u}(r, i-1 \vdash_u \phi)$, it follows $secure_{P,u}(r, i \vdash_u \phi)$.
    We now prove our claim that the execution of t does not alter the content of the tables in $tables(\phi)$. If the trigger is not enabled, the claim is trivial. In the following, we assume the trigger is enabled. Then, t's action is an INSERT statement. This case amount to claiming that the INSERT statement $\langle u', \texttt{INSERT}, R, \overline{t}\rangle$ does not alter the content of the tables in $tables(\phi)$ in case $r, i-1 \vdash_u R(\overline{t})$ holds. We proved the claim above in the *Propagate Forward* **INSERT** *Success - 1* case.
39. *Propagate Forward* **DELETE** *Trigger Action.* The proof of this case is similar to that of *Propagate Forward* **INSERT** *Trigger Action.*
40. *Propagate Backward* **INSERT** *Trigger Action.* The proof of this case is similar to that of *Propagate Forward* **INSERT** *Trigger Action.*
41. *Propagate Backward* **DELETE** *Trigger Action.* The proof of this case is similar to that of *Propagate Forward* **INSERT** *Trigger Action.*
42. *Trigger FD* **INSERT** *Disabled Backward.* Let i be such that $r^i = r^{i-1} \cdot t \cdot s$, where $s = \langle db', U', sec', T', V', c'\rangle \in \Omega_M$, $t \in \mathcal{TRIGGER}_D$, $last(r^{i-1}) = \langle db, U, sec, T, V, c\rangle$, and ϕ be t's actual WHEN condition obtained by replacing all free variables with the values in $tpl(last(r^{i-1}))$. Furthermore, let $act = \langle u', \texttt{INSERT}, R, (\overline{v}, \overline{w}, \overline{q})\rangle$ be t's actual action and α be $\exists \overline{y}, \overline{z}. R(\overline{v}, \overline{y}, \overline{z}) \wedge \overline{y} \neq \overline{w}$. From the rule's definition, it follows that $secEx(s) = \bot$. From this, it follows that $f(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$. From this and f's definition, it follows $f^u_{conf}(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = \top$ since $user(last(r^{i-1}), \langle u', \texttt{SELECT}, \phi\rangle) = u$ since u is t's invoker. From this and f^u_{conf}'s definition, it follows that $secure(u, \neg\phi, last(r^{i-1})) = \top$. From this, it follows that $secure(u, \phi, last(r^{i-1})) = \top$. From this and Lemma F.7, it follows that also $secure_{P,u}(r, i-1 \vdash_u \phi)$.
43. *Trigger ID* **INSERT** *Disabled Backward.* The proof of this case is similar to that of *Trigger FD* **INSERT** *Disabled Backward.*
44. *Trigger ID* **DELETE** *Disabled Backward.* The proof of this case is similar to that of *Trigger FD* **INSERT** *Dis-*

abled Backward.

This completes the proof of the induction step.

This completes the proof. $\square$

In Lemma $G.7$ and Lemma $G.8$, we show that actions and triggers preserve the equivalence class for any LTS that uses f as PDP.

LEMMA G.7. *Let u be a user in $\mathcal{U}$, $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is as above, L be the P-LTS. For any run $r \in traces(L)$ and any action $a \in \mathcal{A}_{D,u}$, if $extend(r, a)$ is defined, then a preserves the equivalence class for r, P, and u.*

PROOF. Let u be a user in $\mathcal{U}$, $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is as above, and L be the P-LTS. In the following, we use e to refer to the *extend* function. We prove our claim by contradiction. Assume, for contradiction's sake, that there is a run $r \in traces(L)$ and an action $a \in \mathcal{A}_{D,u}$ such that $e(r, a)$ is defined and a does not preserve the equivalence class for r, P, and u. According to the LTS semantics, the fact that $e(r, a)$ is defined implies that $triggers(last(r)) = \epsilon$. Therefore, $triggers(last(r')) = \epsilon$ holds as well for any for any $r' \in [\![r]\!]_{P,u}$ (because r and r' are indistinguishable and, therefore, their projections are consistent), and, thus, $e(r', a)$ is defined as well for any $r' \in [\![r]\!]_{P,u}$. There are a number of cases depending on a:

1. $a = \langle u, \texttt{SELECT}, q\rangle$. There are two cases:
   (a) $secEx(last(e(r, a))) = \bot$. From the LTS rules and $secEx(last(e(r, a))) = \bot$, it follows that $f(last(r), a) = \top$. From this and Lemma $G.5$, it follows that $f(last(r'), a) = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows $secEx(last(e(r', a))) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From $f(last(r'), a) = \top$ for any $r' \in [\![r]\!]_{P,u}$, it follows $f_{conf}^{user(last(r'),a)}(last(r'), a) = \top$ for any $r' \in [\![r]\!]_{P,u}$. Note that $user(last(r'), a) = u$ for any $r' \in [\![r]\!]_{P,u}$ because $trigger(last(r')) = \epsilon$ and $u \in \mathcal{A}_{D,u}$. From this, $f_{conf}^{user(last(r'),a)}(last(r'), a) = \top$ for any $r' \in [\![r]\!]_{P,u}$, and f_{conf}^{u}'s definition, it follows that $secure(u, q, last(r')) = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and Lemma $F.7$, it follows that $[q]^{last(r').db} = [q]^{last(r).db}$ for all $r' \in [\![r]\!]_{P,u}$. Furthermore, it follows trivially from the LTS rule ***SELECT** Success*, that the state after a's execution is data indistinguishable from $last(r)$. It is also easy to see that $e(r', a)$ is well-defined for any $r' \in [\![r]\!]_{P,u}$. From the considerations above and $r' \in [\![r]\!]_{P,u}$, it follows trivially that $e(r', a) \in [\![e(r, a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r', a)$. This leads to a contradiction.
   (b) $secEx(last(e(r, a))) = \top$. From the LTS rules and $secEx(last(e(r, a))) = \top$, it follows that $f(last(r), a) = \bot$. From this and Lemma $G.5$, it follows that $f(last(r'), a) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows $secEx(last(e(r', a))) = \top$ for any $r' \in [\![r]\!]_{P,u}$. The data indistinguishability between $last(e(r', a))$ and $last(e(r, a))$ follows trivially from the data indistinguishability between $last(r')$ and $last(r)$. Therefore, for any run $r' \in [\![r]\!]_{P,C}$, there is exactly one run $e(r', a)$. From the considerations above, it follows trivially that $e(r', a) \in [\![e(r, a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r', a)$. This leads to a contradiction.

   Both cases leads to a contradiction. This completes the proof for $a = \langle u, \texttt{SELECT}, q\rangle$.
2. $a = \langle u, \texttt{INSERT}, R, \bar{t}\rangle$. In the following, we denote by gI the function *getInfo*, by gS the function *getInfoS*, and by gV the function *getInfoV*. There are three cases:
   (a) $secEx(last(e(r, a))) = \bot$ and $Ex(last(e(r, a))) = \emptyset$. From the LTS rules and $secEx(last(e(r, a))) = \bot$, it follows that $f(last(r), a) = \top$. From this and Lemma $G.5$, it follows that $f(last(r'), a) = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows that $secEx(last(e(r', a))) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From $f(last(r), a) = \top$, it follows that $f_{conf}^{u}(last(r), a) = \top$ because $user(last(r), a) = u$ since $trigger(last(r), a) = \epsilon$ and $a \in \mathcal{A}_{D,u}$. From this and f_{conf}^{u}'s definition, it follows that $secure(u, gS(\gamma, act), last(r))$ holds for any integrity constraint γ in $Dep(\Gamma, a)$. From $Ex(last(e(r, a))) = \emptyset$ and Proposition $F.7$, it follows $[gS(\gamma, act)]^{last(r).db} = \top$. From this, $secure(u, gS(\gamma, act), last(r))$, and Lemma $F.7$, it follows that $[gS(\gamma, act)]^{last(r').db} = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and Proposition $F.7$, it follows that $Ex(last(e(r', a))) = \emptyset$ for any $r' \in [\![r]\!]_{P,u}$. We claim that, for any $r' \in [\![r]\!]_{P,u}$, $last(e(r, a))$ and $last(e(r', a))$ are data indistinguishable. From this and the above considerations, it follows trivially that $e(r', a) \in [\![e(r, a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r', a)$. This leads to a contradiction.
   We now prove our claim that for any $r' \in [\![r]\!]_{P,u}$, $last(e(r, a))$ and $last(e(r', a))$ are data indistinguishable. We prove the claim by contradiction. Let $s_2 = \langle db_2, U_2, sec_2, T_2, V_2\rangle$ be $pState(last(e(r, a)))$, $s_2' = \langle db_2', U_2', sec_2', T_2', V_2'\rangle$ be $pState(last(e(r', a)))$, $s_1 = \langle db_1, U_1, sec_1, T_1, V_1\rangle$ be $pState(last(r))$, and $s_1' = \langle db_1', U_1', sec_1', T_1', V_1'\rangle$ be $pState(last(r'))$. In the following, we denote the *permissions* function by p. Furthermore, note that s_1 and s_1' are data-indistinguishable because $r' \in [\![r]\!]_{P,u}$. There are a number of cases:
   i. $U_2 \neq U_2'$. Since a is an INSERT operation, it follows that $U_1 = U_2$ and $U_1' = U_2'$. Furthermore, from $s_1 \cong_{M,u}^{data} s_1'$, it follows that $U_1 = U_1'$. Therefore, $U_2 = U_2'$ leading to a contradiction.
   ii. $sec_2 \neq sec_2'$. The proof is similar to the case $U_2 \neq U_2'$.
   iii. $T_2 \neq T_2'$. The proof is similar to the case $U_2 \neq U_2'$.
   iv. $V_2 \neq V_2'$. The proof is similar to the case $U_2 \neq U_2'$.
   v. there is a table R' for which $\langle \oplus, \texttt{SELECT}, R\rangle \in p(s_2, u)$ and $db_2(R') \neq db_2'(R')$. Note that $p(s_2, u) = p(s_1, u)$. There are two cases:
      - $R = R'$. From $s_1 \cong_{M,u}^{data} s_1'$ and $\langle \oplus, \texttt{SELECT}, R\rangle \in p(s_2, u)$, it follows that $db_1(R') = db_1'(R')$. From this and the fact that a has been executed successfully both in $e(r, a)$ and $e(r', a)$, it follows that $db_2(R') = db_1(R') \cup \{\bar{t}\}$ and $db_2'(R') = db_1'(R') \cup \{\bar{t}\}$. From this and $db_1(R') = db_1'(R')$, it follows that $db_2(R') = db_2'(R')$ leading to a contradiction.
      - $R \neq R'$. From $s_1 \cong_{M,u}^{data} s_1'$ and $\langle \oplus, \texttt{SELECT}, R\rangle$

$\in p(s_2, u)$, it follows that $db_1(R') = db'_1(R')$. From this and the fact that a does not modify R', it follows that $db_1(R') = db_2(R')$ and $db'_1(R') = db'_2(R')$. From this and $db_1(R') = db'_1(R')$, it follows that $db_2(R') = db'_2(R')$ leading to a contradiction.

vi. there is a view v for which $\langle \oplus, \texttt{SELECT}, v\rangle \in p(s_2, u)$ and $db_2(v) \neq db'_2(v)$. Note that $p(s_2, u) = p(s_1, u)$. Since a has been successfully executed in both states, we know that $leak(s_1, a, u)$ hold. There are two cases:
- $R \notin tDet(v, s, M)$. Then, $v(s_1) = v(s_2)$ and $v(s'_1) = v(s'_2)$ (because R's content does not determine v's materialization). From $s_1 \cong^{data}_{M,u} s'_1$ and the fact that a modifies only R, it follows that $v(db_2) = v(db'_2)$ leading to a contradiction.
- $R \in tDet(v, s, M)$ and for all $o \in tDet(v, s, M)$, $\langle \oplus, \texttt{SELECT}, o\rangle \in p(s_1, u)$. From this and $s_1 \cong^{data}_{M,u} s'_1$, it follows that, for all $o \in tDet(v, s, M)$, $o(s_1) = o(s'_1)$. If $o \neq R$, $o(s_1) = o(s'_1) = o(s_2) = o(s'_2)$. From $\langle \oplus, \texttt{SELECT}, R\rangle \in p(s_1, u)$ and $s_1 \cong^{data}_{M,u} s'_1$, it follows that $db_1(R) = db'_1(R)$. From this and the fact that a has been executed successfully both in $e(r, a)$ and $e(r', a)$, it follows that $db_2(R) = db_1(R) \cup \{\bar{t}\}$ and $db'_2(R) = db'_1(R) \cup \{\bar{t}\}$. From this and $db_1(R) = db'_1(R)$, it follows that $db_2(R) = db'_2(R)$. From this and for all $o \in tDet(v, s, M)$ such that $o \neq R$, $o(s_2) = o(s'_2)$, it follows that for all $o \in tDet(v, s, M)$, $o(s_2) = o(s'_2)$. Since the content of all tables determining v is the same in s_2 and s'_2, it follows that $db_2(v) = db'_2(v)$ leading to a contradiction.

All the cases lead to a contradiction.

(b) $secEx(last(e(r, a))) = \bot$ and $Ex(last(e(r, a))) \neq \emptyset$. From the LTS rules and $secEx(last(e(r, a))) = \bot$, it follows that $f(last(r), a) = \top$. From this and Lemma $G.5$, it follows that $f(last(r'), a) = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows that $secEx(last(e(r', a))) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. Assume that the exception has been caused by the constraint γ, i.e., $\gamma \in Ex(last(e(r, a)))$. From this and Proposition $F.7$, it follows that $gV(\gamma, a)$ holds in $last(r).db$. From $f(last(r), a) = \top$ and f's definition, it follows that $f^u_{conf}(last(r), a) = \top$ because $user(last(r), a) = u$ since $trigger(last(r)) = \epsilon$ and $a \in \mathcal{A}_{D,u}$. From this and f^u_{conf}'s definition, it follows that $secure(u, gV(\gamma, a), last(r))$ holds. From this, Lemma $F.7$, and $[gV(\gamma, a)]^{last(r).db} = \top$, it follows that also $[gV(\gamma, act)]^{last(r').db} = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and Proposition $F.7$, it follows that $\gamma \in Ex(last(e(r', a)))$ for any $r' \in [\![r]\!]_{P,u}$. The data indistinguishability between $last(e(r, a))$ and $last(e(r', a))$ follows trivially from the data indistinguishability between $last(r)$ and $last(r')$ for any $r' \in [\![r]\!]_{P,u}$. Therefore, for any run $r' \in [\![r]\!]_{P,u}$, there is exactly one run $e(r', a)$. From the considerations above, it follows trivially that $e(r', a) \in [\![e(r, a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r', a)$. This leads to a contradiction.

(c) $secEx(last(e(r, a))) = \top$. From the LTS rules and $secEx(last(e(r, a))) = \top$, it follows that $f(last(r), a) = \bot$. From this and Lemma $G.5$, it follows that $f(last(r'), a) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows $secEx(last(e(r', a))) = \top$ for any $r' \in [\![r]\!]_{P,u}$. The data indistinguishability between $last(e(r, a))$ and $last(e(r', a))$ follows trivially from that between $last(r)$ and $last(r')$ for any $r' \in [\![r]\!]_{P,u}$. Therefore, for any run $r' \in [\![r]\!]_{P,u}$, there is exactly one run $e(r', a)$. From the considerations above, it follows trivially that $e(r', a) \in [\![e(r, a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r', a)$. This leads to a contradiction.

All cases lead to a contradiction. This completes the proof for $a = \langle u, \texttt{INSERT}, R, \bar{t}\rangle$.

3. $a = \langle u, \texttt{DELETE}, R, \bar{t}\rangle$. The proof is similar to that for $a = \langle u, \texttt{INSERT}, R, \bar{t}\rangle$.
4. $a = \langle \oplus, u', p, u\rangle$. There are two cases:
   (a) $secEx(last(e(r, a))) = \bot$. We assume that $p = \langle \texttt{SELECT}, O\rangle$ for some $O \in D \cup V$. If this is not the case, the proof is trivial. Furthermore, we also assume that $u' = u$, otherwise the proof is, again, trivial since the new permission does not influence u's permissions. From the LTS rules and $secEx(last(e(r, a))) = \bot$, it follows that $f(last(r), a) = \top$. From this and Lemma $G.5$, it follows that $f(last(r'), a) = \top$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows that $secEx(last(e(r', a))) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From $secEx(last(e(r', a))) = \bot$ and f^u_{conf}'s definition, it follows that $last(r').sec = last(e(r', a)).sec$. Therefore, since $last(r)$ and $last(r')$ are data indistinguishable, for any $r' \in [\![r]\!]_{P,u}$, then also $last(e(r, a))$ and $last(e(r', a))$ are data indistinguishable. Therefore, for any run $r' \in [\![r]\!]_{P,u}$, there is exactly one run $e(r', a)$. From the considerations above, it follows trivially that $e(r', a) \in [\![e(r, a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r', a)$. This leads to a contradiction.
   (b) $secEx(last(e(r, a))) = \top$. From the LTS rules and $secEx(last(e(r, a))) = \top$, it follows $f(last(r), a) = \bot$. From this and Lemma $G.5$, it follows that $f(last(r'), a) = \bot$ for any $r' \in [\![r]\!]_{P,u}$. From this and the LTS rules, it follows $secEx(last(e(r', a))) = \top$ for any $r' \in [\![r]\!]_{P,u}$. The data indistinguishability between $last(e(r', a))$ and $last(e(r, a))$ follows trivially from the data indistinguishability between $last(r')$ and $last(r)$. Therefore, for any run $r' \in [\![r]\!]_{P,u}$, there is exactly one run $e(r', a)$. From the considerations above, it follows trivially $e(r', a) \in [\![e(r, a)]\!]_{P,u}$. The bijection b is trivially $b(r') = e(r', a)$. This leads to a contradiction.

   Both cases lead to a contradiction. This completes the proof for $a = \langle \oplus, u', p, u\rangle$.
5. $a = \langle \oplus^*, u', p, u\rangle$. The proof is similar to that for $a = \langle \oplus, u', p, u\rangle$.
6. $a = \langle \ominus, u', p, u\rangle$. The proof is similar to that for $a = \langle u, \texttt{SELECT}, q\rangle$. The only difference is in proving that for any $r' \in [\![r]\!]_{P,u}$, $last(e(r, a))$ and $last(e(r', a))$ are data indistinguishable. Assume, for contradiction's sake, that this is not the case. Let $s_2 = \langle db_2, U_2, sec_2, T_2, V_2\rangle$ be $pState(last(e(r, a)))$ and $s'_2 = \langle db'_2, U'_2, sec'_2, T'_2, V'_2\rangle$ be $pState(last(e(r', a)))$. Furthermore, let $s_1 = \langle db_1, U_1, sec_1, T_1, V_1\rangle$ be $pState(last(r))$ and $s'_1 = \langle db'_1, U'_1, sec'_1, T'_1, V'_1\rangle$ be $pState(last(r'))$. In the following, we de-

note the *permissions* function by p. Furthermore, note that s_1 and s'_1 are data-indistinguishable because $r' \in [\![r]\!]_{P,u}$. There are a number of cases:

(a) $U_2 \neq U'_2$. Since a is an `REVOKE` operation, it follows that $U_1 = U_2$ and $U'_1 = U'_2$. Furthermore, from $s_1 \cong^{data}_{u,M} s'_1$, it follows that $U_1 = U'_1$. Therefore, $U_2 = U'_2$ leading to a contradiction.

(b) $sec_2 \neq sec'_2$. From $s_1 \cong^{data}_{u,M} s'_1$, it follows that $sec_1 = sec'_1$. From a's definition and the LTS rules, it follows that $sec_2 = revoke(sec_1, u', p, u)$ and $sec'_2 = revoke(sec'_1, u', p, u)$. From this and $sec_1 = sec'_1$, it follows that $sec_2 = sec'_2$ leading to a contradiction.

(c) $T_2 \neq T'_2$. The proof is similar to the case $U_2 \neq U'_2$.

(d) $V_2 \neq V'_2$. The proof is similar to the case $U_2 \neq U'_2$.

(e) there is a table R for which $\langle \oplus, \texttt{SELECT}, R\rangle \in p(s_2, u)$ and $db_2(R) \neq db'_2(R)$. Since a is an `REVOKE` operation, it follows that $db_1 = db_2$ and $db'_1 = db'_2$. Furthermore, from $s_1 \cong^{data}_{u,M} s'_1$, it follows that $db_1(R) = db'_1(R)$. From this, $db_1 = db_2$, and $db'_1 = db'_2$, it follows that $db_2(R) = db'_2(R)$ leading to a contradiction.

(f) there a view v for which $\langle \oplus, \texttt{SELECT}, v\rangle \in p(s_2, u)$ and $db_2(v) \neq db'_2(v)$. Since a is an `REVOKE` operation, it follows that $db_1 = db_2$ and $db'_1 = db'_2$. Furthermore, from $s_1 \cong^{data}_{u,M} s'_1$, it follows that $db_1(v) = db'_1(v)$. From this, $db_1 = db_2$, and $db'_1 = db'_2$, it follows that $db_2(v) = db'_2(v)$ leading to a contradiction.

All the cases lead to a contradiction.

7. $a = \langle u, \texttt{CREATE}, o\rangle$. The proof is similar to that for $a = \langle \ominus, u', p, u\rangle$.
8. $a = \langle u, \texttt{ADD\_USER}, u'\rangle$. The proof is similar to that for $a = \langle \ominus, u', p, u\rangle$.

This completes the proof. $\square$

LEMMA G.8. *Let u be a user in $\mathcal{U}$, $P = \langle M, f\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f is as above, and L be the P-LTS. For any run $r \in traces(L)$ such that $invoker(last(r)) = u$ and any trigger $t \in \mathcal{TRIGGER}_D$, if $extend(r, t)$ is defined, then t preserves the equivalence class for r, M, and u.*

PROOF. Let u be a user in $\mathcal{U}$, $P = \langle M, f^u_{conf}\rangle$ be an extended configuration, where $M = \langle D, \Gamma\rangle$ is a system configuration and f^u_{conf} is as above, and L be the P-LTS. In the following, we use e to refer to the *extend* function. The proof in cases where the trigger t is not enabled or t's `WHEN` condition is not secure are similar to the proof of the `SELECT` case of Lemma $G.7$. In the following, we therefore assume that the trigger t is enabled and that its `WHEN` condition is secure. We prove our claim by contradiction. Assume, for contradiction's sake, that there is a run $r \in traces(L)$ such that $invoker(last(r)) = u$ and a trigger t such that $e(r, t)$ is defined and t does not preserve the equivalence class for r, P, and u. Since $invoker(last(r)) = u$ and $e(r, t)$ is defined, then $e(r', t)$ is defined as well for any $r' \in [\![r]\!]_{P,u}$ (indeed, from $invoker(last(r)) = u$, it follows that the last action in r is either an action issued by u or a trigger invoker by u. From this, the fact that $e(r, t)$ is defined, and the fact that r and r' are indistinguishable, it follows that $trigger(last(r)) = trigger(last(r')) = t$). Let a be t's action and $w = \langle u', \texttt{SELECT}, q\rangle$ be the `SELECT` command associated with t's `WHEN` condition. Let s be the state $last(r)$, s' be the state obtained just after the execution of the `WHEN` condition, and s'' be the state $last(e(r, t))$. There are a number of cases depending on t's action a:

1. $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$. There are three cases:
   (a) $secEx(last(e(r, a))) = \bot$ and $Ex(last(e(r, a))) = \emptyset$. The proof of this case is similar to that of the corresponding case in Lemma $G.7$.
   (b) $secEx(last(e(r, a))) = \bot$ and $Ex(last(e(r, a))) \neq \emptyset$. The only difference between the proof of this case in this Lemma and in that of Lemma $G.7$ is that we have to establish again the data indistinguishability between $last(e(r, t))$ and $last(e(r', t))$. Indeed, for triggers the roll-back state is, in general, different from the one immediately before the trigger's execution, i.e., it may be that $pState(last(e(r, t))) \neq pState(last(r))$. We now prove that $last(e(r, t))$ and $last(e(r', t))$ are data indistinguishable. From the LTS semantics, it follows that $r = p \cdot s_0 \cdot \langle invoker(last(r)), op, R', \overline{v}\rangle \cdot s_1 \cdot t_1 \cdot \ldots \cdot s_{n-1} \cdot t_n \cdot s_n$, where $p \in traces(L)$ and $t_1, \ldots, t_n \in \mathcal{TRIGGER}_D$. Similarly, $r' = p' \cdot s'_0 \cdot \langle invoker(last(r)), op, R', \overline{v}\rangle \cdot s'_1 \cdot t_1 \cdot \ldots \cdot s'_{n-1} \cdot t_n \cdot s'_n$, where $p' \in traces(L)$, $p \cong_{P,u} p'$, and all states s_i and s'_i are data indistinguishable. Then, the roll-back states are, respectively, s_0 and s'_0, which are data indistinguishable. From the LTS rules, $last(e(r, a)) = s_0$ and $last(e(r', a)) = s'_0$. Therefore, the data indistinguishability between $last(e(r, a))$ and $last(e(r', a))$ follows trivially for any $r' \in [\![r]\!]_{P,u}$.
   (c) $secEx(e(r, a)) = \top$. The proof is similar to the previous case.

   All cases lead to a contradiction. This completes the proof for $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$.
2. $a = \langle u', \texttt{DELETE}, R, \overline{t}\rangle$. The proof is similar to that for $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$.
3. $a = \langle \oplus, u'', p, u'\rangle$. There are two cases:
   (a) $secEx(last(e(r, a))) = \bot$. In this case, the proof is similar to the corresponding case in Lemma $G.7$.
   (b) $secEx(last(e(r, a))) = \top$. The proof is similar to the $secEx(last(e(r, a))) = \top$ case of $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$.

   Both cases lead to a contradiction. This completes the proof for $a = \langle \oplus, u'', p, u'\rangle$.
4. $a = \langle \oplus^*, u'', p, u'\rangle$. The proof is similar to that for $a = \langle \oplus, u'', p, u'\rangle$.
5. $a = \langle \ominus, u'', p, u'\rangle$. The proof is similar to that for $a = \langle u', \texttt{INSERT}, R, \overline{t}\rangle$.

This completes the proof. $\square$

H. DATABASE ACCESS CONTROL AND INFORMATION FLOW CONTROL

Here, we first show that the notion of secure judgment can be seen as an instance of non-interference. Afterwards, we present NI-data confidentiality, a security notion for database access control that is an instance of non-interference. Finally, we show that data confidentiality and NI-data confidentiality are equivalent. For non-interference, we use terminology and notation taken from [28].

It is easy to see that the notion of secure judgment is an instance of non-interference over relational calculus sentences. Indeed, the set of all programs is just the set of all sentences, the set of inputs is the set of all runs, the equivalence relation between the inputs is $\cong_{P,u}$, the set of outputs is $\{\top, \bot\}$, the equivalence relation between the outputs is the equality, and the semantics of the programs is obtained by evaluating the sentences, according to the relational calculus semantics, over the database state in the last state of a run. Using a similar argument, one can easily show that both determinacy [34] and instance-based determinacy [30] are just instances of non-interference over relational calculus sentences.

Before defining NI-data confidentiality, we need some machinery. Let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, $u \in \mathcal{U}$ be a user, $\vdash_u$ be a (P, u)-attacker model, and $\cong$ be a P-indistinguishability relation. Given a run r, we denote by $K(r)$ the set of all formulae that the user u can derive from any extension of r using A, i.e., $\{\phi \in RC_{bool} \mid \exists s \in traces(L), i \in \mathbb{N}.\, s, i \vdash_u \phi \in A \wedge s^{|r|} = r\}$. Moreover, given a set of formulae K, we say that two runs r and r' *agree* on K, denoted by $r \equiv_K r'$, iff for all $\phi \in K$, ϕ holds in the last states of r and r'. Given a system state $s = \langle db, U, sec, T, V, c\rangle$, we denote by $s.db$ the database state db.

We are now ready to define NI-data confidentiality notion.

Definition H.1. Let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, $u \in \mathcal{U}$ be a user, A be a (P, u)-attacker model, and $\cong$ be a P-indistinguishability relation. We say that f *provides NI-data confidentiality with respect to P, u, A, and* $\cong$ iff for all runs $r, r' \in traces(L)$, if $r \cong r'$ holds, then $r \equiv_{K(r)\cup K(r')} r'$ holds. $\square$

Finally, we prove that NI-data confidentiality and data confidentiality are equivalent.

PROPOSITION H.1. *Let $P = \langle M, f\rangle$ be an extended configuration, L be the P-LTS, $u \in \mathcal{U}$ be a user, $\vdash_u$ be a (P, u)-attacker model, and $\cong_{P,u}$ be a (P, u)-indistinguishability relation. The PDP f provides data confidentiality iff it provides NI-data confidentiality.*

PROOF. We prove the two directions separately.

($\Rightarrow$) We prove this direction by contradiction. Assume that f provides data confidentiality but it does not provide NI-data confidentiality. From the fact that NI-data confidentiality does not hold, it follows that there are two runs $r, r' \in traces(L)$ such that $r \cong r'$ but $r \not\equiv_{K(r)\cup K(r')} r'$. From $r \not\equiv_{K(r)\cup K(r')} r'$, it follows that there are two cases:

1. there is a run $s \in traces(L)$ such that $s^{|r|} = r$, $s, |r| \vdash_u \phi \in A$, and $[\phi]^{last(r).db} \neq [\phi]^{last(r').db}$. From this, it follows that $secure_{P,\cong}(s, |r| \vdash_u \phi)$ does not hold, since $s^{|r|} = r$, $[\phi]^{last(r).db} \neq [\phi]^{last(r').db}$, and $r \cong r'$. This contradicts the fact that f provides data confidentiality.
2. there is a run $s \in traces(L)$ such that $s^{|r'|} = r'$, $s, |r'| \vdash_u \phi \in A$, and $[\phi]^{last(r).db} \neq [\phi]^{last(r').db}$. From this, it follows that $secure_{P,\cong}(s, |r'| \vdash_u \phi)$ does not hold, that is not secure, since $s^{|r'|} = r'$, $[\phi]^{last(r).db} \neq [\phi]^{last(r').db}$, and $r \cong r'$. This contradicts the fact that f provides data confidentiality.

Since both cases lead to a contradiction, this concludes the proof of this direction.

($\Leftarrow$) We prove this direction by contradiction. Assume that f provides NI-data confidentiality but it does not provide data confidentiality. From the fact that data confidentiality does not hold, it follows that there is a runs $r \in traces(L)$, an index i, and a sentence ϕ such that $r, i \vdash_u \phi \in A$ and $secure_{P,\cong}(r, i \vdash_u \phi)$ does not hold. From this and $secure_{P,\cong}$ $(r, i \vdash_u \phi)$'s definition, it follows that there are two runs $r, r' \in traces(L)$, an index i, and a sentence ϕ such that $r, i \vdash_u \phi \in A$, $r^i \cong r'$, and $[\phi]^{last(r^i).db} \neq [\phi]^{last(r').db}$. From this and $|r^i| = i$, it follows that there are two runs $r, r' \in traces(L)$ and a sentence ϕ such that $r, |r^i| \vdash_u \phi \in A$, $r^i \cong r'$, and $[\phi]^{last(r^i).db} \neq [\phi]^{last(r').db}$. By renaming r^i as k and by considering the fact that r is, by definition, an extension of k, it follows that there are two runs $r, r' \in traces(L)$ and a sentence ϕ such that $r, |k| \vdash_u \phi \in A$, $r^{|k|} = k$, $k \cong r'$, and $[\phi]^{last(k).db} \neq [\phi]^{last(r').db}$. From this and $K(k)$'s definition, it follows that there are two runs $k, r' \in traces(L)$ and a sentence ϕ such that $\phi \in K(k)$, $k \cong r'$, and $[\phi]^{last(k).db} \neq [\phi]^{last(r').db}$. From this and $\phi \in K(k)$, it follows that there are two runs $k, r' \in traces(L)$ and a sentence ϕ such that $k \cong r'$ and $k \not\equiv_{K(k)} r'$. From this, it follows that there are two runs $k, r' \in traces(L)$ and a sentence ϕ such that $k \cong_{P,u} r'$, and $k \not\equiv_{K(k)\cup K(r')} r'$. This contradicts the fact that f provides NI-data confidentiality. $\square$

We now show that NI-data confidentiality can be seen as an instance of non-interference. Let M be a system configuration and u be a user. The set of programs $\mathcal{P}$ is the set of all pairs of the form $(f, \vdash_u)$, where f is a system configuration and $\vdash_u$ is a $(\langle M, f\rangle, u)$-attacker model. The set of inputs $\mathcal{I}$ is the set $\{(s, evs) \mid s \in \mathcal{I}_M \wedge evs \in (\mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D)^*\}$. The set of outputs $\mathcal{O}$ is the set of all possible sequences of M-states and labels in $\mathcal{A}_{D,\mathcal{U}} \cup \mathcal{TRIGGER}_D$. The semantics of the programs $\sigma : \mathcal{P} \times \mathcal{I} \to (\mathcal{O} \cup \{\bot\})$ is a total function defined as follows: $\sigma((f, \vdash_u), (s, evs)) = r$ iff (1) r is a run in $traces(L)$, where L is the $\langle M, f\rangle$-LTS, (2) r starts from the state s, and (3) the labels of r are equivalent to evs; $\sigma((f, \vdash_u), (s, evs)) = \bot$ otherwise. Finally, the relation $\sim$ over the set $\mathcal{I}$ is $\sim = \mathcal{I} \times \mathcal{I}$, i.e., any two inputs are indistinguishable, whereas the relation $\equiv$ over the set $\mathcal{O}$ is as follows: for any two $r, r' \in \mathcal{O}$, $r \equiv r'$ iff (1) $r = \bot$, (2) $r' = \bot$, or (3) $r \neq \bot$, $r' \neq \bot$, and if $r \cong_{P,u} r'$, then $r \equiv_{K(r)\cup K(r')} r'$. Note that $\equiv$ is not an equivalence relation, i.e., it is reflexive and symmetric but it is not transitive. Therefore, a PDP f provides NI-data confidentiality (and, therefore, data confidentiality) with respect to an attacker model $\vdash_u$ iff $(f, \vdash_u)$ satisfies non-interferences, where $\mathcal{P}$, $\mathcal{I}$, $\mathcal{O}$, σ, $\sim$, and $\equiv$ are as above.

```
SqlStmt := SelectStmt | SqlBasicStmt | CreateTrigger | CreateView
SqlBasicStmt := InsertStmt | DeleteStmt | GrantStmt | RevokeStmt
SelectStmt := "SELECT DISTINCT" columnList "FROM" tableList "WHERE" expr
columnList := columnId | columnList "," columnId
tableList := tableId | tableList "," tableId
expr := varId "=" const | varId "=" varId | "NOT" "("expr")" | expr ("AND"|"OR") expr |
        "EXISTS" "("SelectStmt")"
InsertStmt := "INSERT INTO" tableId "VALUES ("valueList")"
valueList := const | valueList "," const
DeleteStmt := "DELETE FROM" tableId "WHERE" restrictedExpr
restrictedExpr := varId "=" const | restrictedExpr "AND" varId "=" const
GrantStmt := "GRANT" privilege "TO" userId ("WITH GRANT OPTION")
RevokeStmt := "REVOKE" privilege "FROM" userId "WITH CASCADE"
privilege := "SELECT ON" (tableId | viewId) | "CREATE VIEW" |
        ( "INSERT" | "DELETE" | "CREATE TRIGGER" ) "ON" tableId
CreateTrigger := "CREATE TRIGGER" triggerId "AFTER" ("INSERT" | "DELETE") "ON" tableId
        ("SECURITY DEFINER" | "SECURITY INVOKER") SqlBasicStmt
CreateView := "CREATE VIEW" viewId ("SECURITY DEFINER" | "SECURITY INVOKER")
        AS SelectStmt
```

Figure 41: This is the syntax of the SQL fragment that corresponds to the features we support in this paper.