algorithms

MDPI

Article

Simplified Integrity Checking for an Expressive Class of Denial Constraints

Davide Martinenghi

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo 32, 20133 Milan, Italy; davide.martinenghi@polimi.it

Abstract: Data integrity is crucial for ensuring data correctness and quality and is maintained through integrity constraints that must be continuously checked, especially in data-intensive systems like OLTP. While DBMSs handle very simple cases of constraints (such as primary key and foreign key constraints) well, more complex constraints often require ad hoc solutions. Research since the 1980s has focused on automatic and simplified integrity constraint checking, leveraging the assumption that databases are consistent before updates. This paper presents program transformation operators to generate simplified integrity constraints, focusing on complex constraints expressed in denial form. In particular, we target a class of integrity constraints, called extended denials, which are more general than tuple-generating dependencies and equality-generating dependencies. One of the main contributions of this study consists in the automatic treatment of such a general class of constraints, encompassing the all the most useful and common cases of constraints adopted in practice. Another contribution is the applicability of the proposed technique with a "preventive" approach; unlike all other methods for integrity maintenance, we check whether an update will violate the constraints *before* executing it, so we never have to undo any work, with potentially huge savings in terms of execution overhead. These techniques can be readily applied to standard database practices and can be directly translated into SQL.

Keywords: integrity constraints; simplification; denials; dependencies

check for updates

Academic Editor: Frank Werner

Received: 13 January 2025
Revised: 8 February 2025
Accepted: 18 February 2025
Published: 20 February 2025

Citation: Martinenghi, D. Simplified Integrity Checking for an Expressive Class of Denial Constraints. *Algorithms* **2025**, *18*, 123. https://doi.org/10.3390/a18030123

Copyright: © 2025 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Data integrity is a ubiquitous concern that is at the basis of crucial objectives such as data correctness and data quality. Typically, a specification of the correct states of a database is given through integrity constraints, which can be thought of as formulas that need to be kept satisfied throughout the life of the database. In any data-intensive scenario, such as an OLTP system, data are updated all the time and, therefore, integrity constraints need to be checked continuously. While DBMSs are able to optimally handle common constraints such as key and foreign key constraints, the standard practice is to resort to ad hoc solutions (for instance, at the application level or through manually designed triggers) when the integrity constraints to be maintained are more complex. Automatic integrity constraint checking—and simplified integrity constraint checking in particular—is a topic that has spurred numerous research attempts since the 1980s [1–13] with the objective of mechanically producing the best possible set of conditions to check against a database in the face of updates. The main idea, common to the majority of the approaches, is to exploit the assumption that a database is *consistent* (i.e., satisfies the integrity constraints) before an update; with this, the formula to be checked, corresponding to the original

integrity constraints, may be simplified so as to avoid checks of subformulas that are already guaranteed to hold thanks to the consistency assumption.

For instance, consider the relation $b(I, T)$, indicating that there is a book with an ISBN I and a title T. The fact that the ISBN univocally identifies a book can be expressed as the integrity constraint $T_1 = T_2 \leftarrow b(I, T_1) \wedge b(I, T_2)$ (with all variables implicitly universally quantified). Now, assume that the tuple $b(i, t)$ is added to the database. Under the consistency assumption, one just needs to check that there is no other book with the same ISBN i and a different title than t (we are using set semantics, so having more than one occurrence of $b(i, t)$ is harmless), which could be expressed by the simplified formula $T = t \leftarrow b(i, T)$. Without the consistency assumption, instead, we would have had to check for satisfaction of the original constraint for all the books already present in the database. The difference between the simplified formula and the original one bears heavy implications from the point of view of the complexity of the corresponding check, which, in turn, may depend on the way in which the data are physically organized in the database. In the absence of indices, the simplified formula requires a linear scan of all the books; the original formula, instead, requires checking that, for *every* book present in the database, there is no other book with the same ISBN and a different title, which entails a quadratic (i.e., much higher) cost in the size of the table. With indices (say, a hash-based index on the ISBN), the original check would simply require a linear scan of the books, as an ISBN-based lookup would then take constant time; however, with the same index, the simplified formula would no longer require a linear scan, because a single (constant-time, thus much faster) lookup of the hash-based index would suffice.

Overall, simplified integrity checking aims to maintain satisfaction of the integrity constraints (specified at database schema design time) after each update to the data without overburdening the system with checks that would be too long to carry out. So, "simplification" in this sense means to obtain tests that are simpler to verify than the original constraints, while achieving the same verification effects. As rooted in the defining properties of databases, integrity constraints are one of the building blocks of relational technology and their maintenance is one of the core aspects of DBMSs, which have to guarantee integrity constraint satisfaction at all times.

In this paper, we discuss the use of program transformation operators to the generation of simplified integrity constraints. We implement these operators in terms of rewrite rules based on resolution, subsumption, and replacement of specific patterns. In particular, one of the main contributions of this study is the broad applicability of these operators to a large class of integrity constraints that encompasses all the most common dependencies that are used in standard database design practice, including tuple-generating dependencies (TGDs) and equality-generating dependencies (EGDs), which we express in denial form with negated existential quantifiers, possibly nested.

Another important contribution is the "preventive" perspective of the proposed technique. Traditional methods for simplified integrity checking work as follows: first, a pending update is executed on the database; then, the simplified check is tested against the updated database; and, if the test fails, the update needs to be retracted (typically, with a rollback of the enclosing transaction). Unlike all such methods, we advocate an approach that checks constraint violation *before* the execution of the pending update. This means that, if we detect that the update will harm consistency, we will not even execute it, thereby saving a significant amount of unneeded work consisting of the execution and subsequent retraction of an update.

The techniques described here can be immediately applied to standard database practice, since our logical notation can be easily translated into SQL. In particular, denials correspond to queries whose result must be empty and their maintenance could be enforced

through native database constructs such as triggers [13], whose importance is gaining momentum even outside the scope of relational databases [14].

Our results build on the operators informally presented in [15]. To the best of our knowledge, there is no other approach to date that has considered such an expressive class of constraints for the problem of simplified integrity constraint checking.

Outline. After reviewing related work in Section 2, we introduce some preliminary technical notions in Section 3 in order to set the notation and terminology commonly adopted in relational and deductive databases. We formally introduce the main operators used in the integrity constraint simplification process in Section 4. The core of this proposal is presented in Section 5, where we extend the scope of our operators to a very expressive class of integrity constraints that we call extended denials, for which we offer a comparison with other methods discussed in the pertinent literature along with practical considerations showing how to integrate the procedure into practical systems. Finally, we discuss the extent of our results and provide some concluding remarks in Section 6.

2. Related Work

The idea of simplifying integrity constraints has been long recognized, dating back to at least [1,2], as discussed in the previous section. We continued this line of research in [15,16]. In particular, in [15], we considered an expressive language, $\mathcal{L}_{\text{H}}$, for formulating schemata and constraints, for which we could provide a sound and terminating simplification procedure. An important distinction between our work and other simplification techniques is the "preventive" approach that we adopt; we check whether an update is legal before executing it, so that no work needs to be undone if an illegal update is detected. All other approaches in the literature adopt a "corrective" approach instead, i.e., the update is always executed, then the updated databases is checked for integrity violations, and, if needed, the update is undone. Evidently, a preventive approach allows one to avoid a lot of unneeded work in case of illegal updates (their execution and the subsequent retraction).

The simplification problem is also tightly connected to query containment (QC) [17], i.e., the problem of establishing whether, given two queries q_1 and q_2, the answer to q_1 is always a subset of the answer to q_2 for all database instances. Indeed, "perfect" simplifications may be generated if and only if QC is decidable. Unfortunately, QC is already undecidable for DATALOG databases without negation [18,19].

The kind of integrity constraints considered here are the so-called *static* integrity constraints, in that they refer to properties that must be met by the data in each database state. *Dynamic* constraints, instead, are used to impose restrictions on the way the database states can evolve over time, be it on successive states or on arbitrary sets of states. Dynamic constraints have been considered, e.g., in [20,21].

Another important distinction can be made between *hard* (or *strong*) constraints and *soft* (or *deontic*) constraints. The former ones are used to model necessary requirements of the world that the database represents, like the constraints studied here. Deontic constraints govern what is obligatory but not necessary of the world, so violations of deontic constraints correspond to violations of obligations that have to be reported to the user or administrator of the database rather than inconsistencies proper. The wording "soft constraint" sometimes refers to a condition that should preferably be satisfied, but may also be violated. Deontic constraints have been considered, e.g., in [22] and soft constraints in [23]. Soft constraints are akin to preferences, covered by an abundant literature [24–30].

Other kinds of constraints, of a more structural nature, are the so-called access constraints, or access patterns, whose interaction with integrity constraints proper has also been studied [31].

Often, integrity constraints are characterized as *dependencies*. Many common dependencies are used in database practice. Among those involving a single relation, we mention *functional dependencies* [32] (including *key dependencies*) and *multi-valued dependencies* [33,34]. For the inter-relation case, the most common ones are *inclusion dependencies* (i.e., *referential constraints*). All of these can straightforwardly be represented as extended denials. In particular, many authors acknowledge tuple-generating dependencies (TGDs) and equality-generating dependencies (EGDs) to be the most important types of dependencies in a database, since they encompass most others [35] and are also commonly used for schema mappings in data exchange [36]. The dependencies can be expressed by our extended denials, as we will show in Section 5. So our framework captures the most representative cases of constraints used in the literature.

More detailed classifications of integrity constraints for deductive databases have been attempted by several authors, e.g., [37–39].

Once illegal updates are detected, it must be decided how to restore a consistent database state. We targeted a preventive approach that avoids illegal updates completely. Other approaches, instead, prefer to restore consistency via corrective actions after an illegal update—typically a rollback—but several works compute a *repair* that changes, adds, or deletes tuples of the database in order to satisfy the integrity constraints again. The generation of repairs is a nontrivial issue; see, e.g., [40–44] for surveys on the topic.

In some scenarios, a temporary violation of integrity constraints may be accepted provided that consistency is quickly repaired; if, by nature of the database application, the data are particularly unreliable, inconsistencies may be even considered unavoidable. Approaches that cope with the presence of inconsistencies have been studied and gave rise to *inconsistency-tolerant integrity checking* [45–47]. Besides being checked and tolerated, inconsistency can also be measured, and numerous indicators have been studied to address this problem [48–52].

An orthogonal research avenue is that of allowing inconsistencies to occur in databases but to filter the data during query processing so as to provide a *consistent query answer* [40], i.e., the set of tuples that answer a query in all possible repairs (without, of course, actually having to compute all the repairs). This, however, has repercussions on the complexity of query answering.

Integrity checking has been applied to a number of different contexts, including data integration [53], the presence of aggregates [54,55], the interaction with transaction management [56], the use of symbolic constraints [57], approximate constraints [58], big data [59], and data clouds [60].

We also observe that integrity checking is also commonly included as a typical part of complex data preparation pipelines for subsequent processing based, e.g., on machine learning or clustering algorithms [61–64] on data collected from various sources, such as RFID [65,66], pattern mining [67], crowdsourcing applications [68–70], and streaming data [71].

More recent works in integrity checking have covered a wide spectrum of application contexts, including a blockchain-based data integrity checking [72], possibly to detect insider attacks in DBMSs [73], privacy-preserving integrity checking for the cloud [74], and inconsistency management through repairs for databases under universal constraints [75], a form of integrity constraints that also captures TGDs and EGDs. However, the adopted perspective is that of tolerating and repairing inconsistencies rather than preventing their onset, as is done here.

3. Relational and Deductive Databases

In this chapter, we present the basic notation related to deductive databases, and use the syntax of the DATALOG language as a basis to formulate the main concepts. We refer to standard texts in logic, logic programming, and databases such as [37,38,76] for further discussion. As a notational convention, we use lowercase letters to denote *predicates* (p, q, ...) and *constants* (a, b, ...) and uppercase letters (X, Y, ...) to denote *variables*. A *term* is either a variable or a constant; conventionally, terms are also denoted by lowercase letters (t, s, ...) and sequences of terms are indicated by vector notation, e.g., $\overrightarrow{t}$. The language also includes standard connectives ($\wedge$, $\vee$, $\neg$, $\leftarrow$), quantifiers, and punctuation.

In quantified formulas such as $(\forall XF)$ and $(\exists XF)$, X is said to be *bound*; a variable not bound is said to be *free*. A formula with no free variables is said to be *closed*. Free variables are indicated in boldface ($\mathbf{a}$, $\mathbf{b}$, ...) and referred to as *parameters*; the other variables, unless otherwise stated, are assumed to be universally quantified.

Formulas of the form $p(t_1, \ldots, t_n)$ are called *atoms*. An atom preceded by a $\neg$ symbol is a *negated atom*; a *literal* is either an atom or a negated atom.

Predicates are divided into three pairwise disjoint sets: *intensional*, *extensional*, and *built-in* predicates. Intensional and extensional predicates are collectively called *database predicates*; atoms and literals are classified similarly according to their predicate symbol. There is one built-in binary predicate for term equality ($=$), written using infix notation; $t_1 \neq t_2$ is a shorthand for $\neg(t_1 = t_2)$ for any two terms t_1, t_2.

A *substitution* $\{\overrightarrow{X} / \overrightarrow{t}\}$ is a mapping from the variables in $\overrightarrow{X}$ to the terms in $\overrightarrow{t}$; its application is the expression arising when each occurrence of a variable in $\overrightarrow{X}$ is simultaneously replaced by the corresponding term in $\overrightarrow{t}$. A formula or term which contains no variables is called *ground*. A substitution $\{\overrightarrow{X} / \overrightarrow{Y}\}$ is called a *renaming* if $\overrightarrow{Y}$ is a permutation of $\overrightarrow{X}$. Formulas F, G are *variants* of one another if $F = G\rho$ for some renaming ρ. A substitution σ is said to be more general than θ if there exists a substitution η such that $\theta = \sigma\eta$. A *unifier* of $t_1, \ldots, t_n$ is a substitution σ such that $t_1\sigma = \ldots = t_n\sigma$; σ is a *most general unifier* (mgu) of $t_1, \ldots, t_n$ if it is more general than any other unifier thereof.

A *clause* is a disjunction of literals $L_1 \vee \ldots \vee L_n$. In the context of deductive databases, clauses are expressed in implicational form, $A \leftarrow L_1 \wedge \ldots \wedge L_n$, where A is an atom and $L_1, \ldots, L_n$ are literals; A is called the *head* and $L_1 \wedge \ldots \wedge L_n$ the *body* of the clause. The head is optional and, when it is omitted, the clause is called a *denial*. A *rule* is a clause whose head is intensional, and a *fact* is a clause whose head is extensional and ground and whose body is empty (understood as *true*). Two clauses are *standardized apart* if they have no bound variable in common. As customary, we also assume clauses to be *range-restricted*, i.e., each variable must occur in a positive database literal in the body.

Deductive databases are characterized by three components: facts, rules, and integrity constraints. An *integrity constraint* can, in general, be any (closed) formula. In the context of deductive databases, however, it is customary to express integrity constraints in some canonical form; we adopt here the denial form, which gives a clear indication of what must not occur in the database.

Definition 1 (Schema, database)**.** *A database* schema *S is a pair* $\langle IDB, IC \rangle$*, where IDB (the* intensional database*) is a finite set of range-restricted rules, and IC (the* constraint theory*) is a finite set of denials. A* database *D on S is a pair* $\langle IDB, EDB \rangle$*, where EDB (the* extensional database*) is a finite set of facts; D is said to be* based on *IDB.*

If the *IDB* is understood, the database is identified with *EDB* and the schema with *IC*.

For the semantics of a deductive database, we refer to standard texts, such as [77]. There are database classes for which the existence of a unique *intended* minimal model is guaranteed. One such class is that of *stratified databases*, i.e., those in which no predicate

depends negatively on itself. This can be checked syntactically by building a graph with a node for each predicate and an arc between p and q each time p is in the body and q in the head of the same clause; the arc is labeled with a "$-$" if p occurs negated. The database is stratified if the graph has no cycle including arcs labeled with a "$-$".

Stratified databases admit a unique "intended" minimal model, called the *standard model* [78,79]. Other minimal models are known, e.g., the *perfect model* for *locally stratified deductive databases* [80], the *stable model* semantics [81], and the *well-founded model* semantics [82]. We adopt the semantics of stratified databases and write $D \models \phi$, where D is a database and ϕ a closed formula, to indicate that ϕ holds in D's standard model.

We now introduce a notation for querying and updating a database. A *query* is an expression of the form $\Leftarrow A$, where A is an atom whose predicate is intensional. When no ambiguity arises, a given query may be indicated directly by means of its *defining formula* (instead of the predicate name).

Definition 2 (Update). *An* update *for an extensional predicate p in a database D is an expression of the form $p(\overrightarrow{X}) \Leftarrow p'(\overrightarrow{X})$ where $\Leftarrow p'(\overrightarrow{X})$ is a query for D. For an update U, the* updated *database D^U has the same EDB as D but in which, for every extensional predicate p updated as $p(\overrightarrow{X}) \Leftarrow p'(\overrightarrow{X})$ in U, the subset $\{p(\overrightarrow{t}) \mid D \models p(\overrightarrow{t})\}$ of EDB is replaced by the set $\{p(\overrightarrow{t}) \mid D \models p'(\overrightarrow{t})\}$.*

Example 1. *Consider $IDB_1 = \{p'(X) \leftarrow p(X), \quad p'(X) \leftarrow X = a\}$. The following update U_1 describes the addition of fact $p(a)$: $\{p(X) \Leftarrow p'(X)\}$. For convenience, wherever possible, defining formulas instead of queries will be written in the body of predicate updates. Update U_1 will, e.g., be indicated as follows: $\{p(X) \Leftarrow p(X) \vee X = a\}$.*

The constraint verification problem may be formulated as follows. Given a database D, a constraint theory Γ such that $D \models \Gamma$, and an update U, does $D^U \models \Gamma$ hold too?

Since checking directly whether $D^U \models \Gamma$ holds may be too expensive, we aim to obtain a constraint theory Γ^U, called a *pre-test*, such that $D^U \models \Gamma$ if $D \models \Gamma^U$ and Γ^U is easier to evaluate than Γ.

4. A Simplification Procedure for Obtaining a Pre-Test

We report here the main steps of the simplification procedure described in [15], whose operators will be used as building blocks in the next section to handle an expressive class of integrity constraints.

The procedure consists of two main steps, After and Optimize, which, together, are the main components of the simplification procedure Simp, as sketchily described in Figure 1. The former receives a schema S and an update U, and builds a schema S' that, while applying to the current state, represents the given integrity constraints in the state after the update U. The latter eliminates redundancies from After's output S' and exploits the consistency assumption, i.e., that the integrity constraints in S hold in the state before update U, so as to emit a simplified schema S^*.

We observe that these operators work at the program level instead of the data level, so they will target program specifications (in datalog) rather than datasets. In the experiments and examples discussed here and in Section 5, we shall indeed tests our operators against specifications of integrity constraints (most of which are taken from the cited literature).

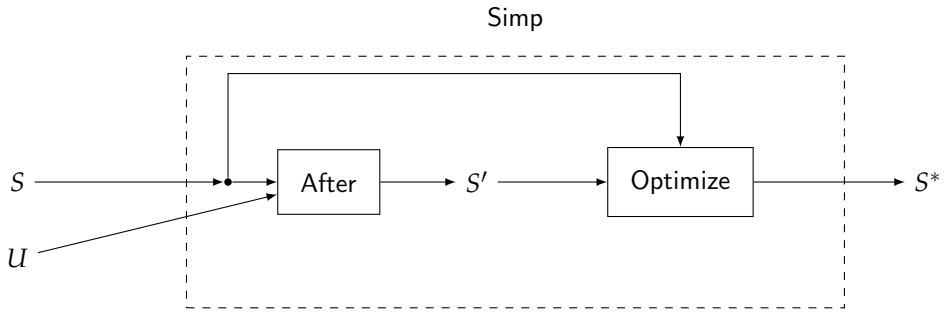

Figure 1. Sketch of the main blocks in the simplification procedure Simp.

For easier reference, Table 1 summarizes the main operators and symbols defined in this section, along with a synthetic explanation of how they work.

Table 1. Operators used for the simplification procedure, as defined in Section 4.

Notation	Explanation
$S' = \mathsf{After}^U(S)$	Schema S' is a WP of schema S wrt update U (Definition 7)
$\mathcal{L}_s$	A class of schemata that can be unfolded as a set of denials (Definition 6)
$\Gamma = \mathsf{Unfold}_{\mathcal{L}_s}(S)$	For a schema $S = \langle IDB, IC\rangle \in \mathcal{L}_s$, Γ is the unfolding of IC wrt IDB (Definition 8)
$\Gamma = \mathsf{After}^U_{\mathcal{L}_s}(S)$	The set of denials Γ is a WP of schema $S \in \mathcal{L}_s$ wrt update U, and is defined as $\Gamma = \mathsf{Unfold}_{\mathcal{L}_s}(\mathsf{After}^U(S))$
$\phi_1 \sqsubseteq \phi_2$	Denial ϕ_1 subsumes denial ϕ_2 (Definition 9)
$\phi_1 \sqsubset \phi_2$	Denial ϕ_1 strictly subsumes denial ϕ_2 (Definition 9)
ϕ^-	Reduction: same as denial ϕ but without redundancies (Definition 10)
ϕ^+, Φ^+	Expansion: same as denial ϕ (or set of denials Φ) with constants and repeated variables replaced by new variables and equalities; see [8]
$\Gamma \vdash_R \phi$	Denial ϕ is derived from the set of denials Γ. In particular, there is a resolution derivation of a denial ψ from Γ^+ such that $\psi^- \sqsubseteq \phi$ (Definition 11)
$\Gamma' = \mathsf{Optimize}^{\Delta}_{\mathcal{L}_s}(\Gamma)$	Γ' is the result of applying denial elimination and literal elimination from the set of denials $\Gamma \in \mathcal{L}_s$ as long as possible by trusting the set of denials $\Delta \in \mathcal{L}_s$ (Definition 12)
$\Phi = \mathsf{Simp}^U_{\mathcal{L}_s}(S)$	The set of denials Φ is a CWP of schema $S \in \mathcal{L}_s$ wrt update U computed through $\mathsf{Optimize}_{\mathcal{L}_s}$ and $\mathsf{After}_{\mathcal{L}_s}$ (Definition 13)

A mere condition for checking integrity without actually executing the update is given by the notion of weakest precondition [6,83,84].

Definition 3 (Weakest precondition)**.** *Let $S = \langle IDB, \Gamma\rangle$ be a schema and U an update. A schema $S' = \langle IDB', \Gamma'\rangle$ is a* weakest precondition (WP) *of S with respect to U whenever $D \models \Gamma' \Leftrightarrow D^U \models \Gamma$ for any database D based on $IDB \cup IDB'$.*

To employ the consistency assumption, we extend the class of checks of interest as follows.

Definition 4 (Conditional weakest precondition)**.** *Let $S = \langle IDB, \Gamma\rangle$ be a schema and U an update. Schema $S' = \langle IDB', \Gamma'\rangle$ is a* conditional weakest precondition (CWP) *of S wrt U whenever $D \models \Gamma' \Leftrightarrow D^U \models \Gamma$ for any database D such that $D \models \Gamma$.*

A WP is also a CWP but the reverse does not necessarily hold.

Example 2. *Consider the update and IDB from Example 1 and let* Γ_1 *be the constraint theory* $\{\leftarrow p(X) \wedge q(X)\}$ *stating that p and q are mutually exclusive. Then,* $\{\leftarrow q(a)\}$ *is a CWP (but not a WP) of* Γ_1 *with respect to* U_1.

4.1. The Core Language $\mathcal{L}_s$

We now formally characterize a language, that we call $\mathcal{L}_s$, on which a core simplification procedure can be applied. We introduce a few technical notions that are needed in order to precisely identify the class of predicates, integrity constraints, and updates that are part of $\mathcal{L}_s$, which is a non-recursive, function-free language equipped with negation.

Definition 5 (Starred dependency graph)**.** *Let* $S = \langle IDB, \Gamma \rangle$ *be a schema in which the IDB consists of a set of disjunctive (range-restricted) predicate definitions and* Γ *is a set of range-restricted denials. Let* $\mathcal{G}$ *be a graph that has a node* N^p *for each predicate p in S plus another node named* $\perp$*, and no other node; if p's defining formula has variables not occurring in the head, then* N^p *is marked with a "*". For any two predicates p and* p' *in S,* $\mathcal{G}$ *has an arc from* N^p *to* $N^{p'}$ *for each occurrence of p in an IDB rule in which* p' *occurs in the head; similarly, there is an arc from* N^p *to* $\perp$ *for each occurrence of p in the body of a denial in* Γ*. In both cases, the arc is labelled with a "*$-$*" (and said to be* negative*) if p occurs negatively.* $\mathcal{G}$ *is the* starred dependency graph *for S.*

Example 3. *Let* $S_1 = \langle IDB_1, \Gamma_1 \rangle$ *and* $S_2 = \langle IDB_2, \Gamma_2 \rangle$ *be schemata with*

$$\begin{aligned} IDB_1 &= \{ \quad s_1(X) \leftarrow r_1(X,Y)\}, \\ \Gamma_1 &= \{ \quad \leftarrow p_1(X) \wedge \neg s_1(X)\}, \\ IDB_2 &= \{ \quad s_2(X) \leftarrow r_2(X,Y), \\ &\qquad\quad q_2(X) \leftarrow \neg s_2(X) \wedge t_2(X)\}, \\ \Gamma_2 &= \{ \quad \leftarrow p_2(X) \wedge \neg q_2(X)\}. \end{aligned}$$

The starred dependency graphs of S_1 *and* S_2 *are shown in Figure 2.*

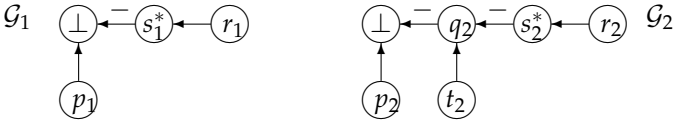

Figure 2. The starred dependency graphs for the schemata of Example 3.

Definition 6 ($\mathcal{L}_s$)**.** *A schema S is in* $\mathcal{L}_s$ *if its starred dependency graph* $\mathcal{G}$ *is acyclic and, in every path in* $\mathcal{G}$ *from a starred node to* $\perp$*, the number of arcs labelled with "*$-$*" is even. An update* $\{p_1(\vec{X}_1) \Leftarrow q_1(\vec{X}_1),\ \ldots,\ p_m(\vec{X}_m) \Leftarrow q_m(\vec{X}_m)\}$ *is in* $\mathcal{L}_s$ *if the graph obtained from* $\mathcal{G}$ *by adding an arc from* N^{q_i} *to* N^{p_i}*,* $1 \leq i \leq m$ *meets the above condition.*

Acyclicity corresponds to absence of recursion. The second condition requires that the unfolding of the intensional predicates in the constraint theory does not introduce any negated existentially quantified variable. In particular, starred nodes correspond to literals with existentially quantified variable (called *non-distinguished*). In order to avoid negated existential quantifiers, the number of "$-$" signs must be even, as an even number of negations means no negation. This requirement will be removed in the next section.

Example 4. *Consider* S_1 *and* S_2 *from Example 3. Clearly,* S_1 *is not in* $\mathcal{L}_s$*, as in its starred dependency graph* $\mathcal{G}_1$ *in Figure 2, there is a path from* s_1^* *to* $\perp$ *containing one "*$-$*" arc, while* S_2 *is in* $\mathcal{L}_s$*, as in* $\mathcal{G}_2$*, the only path from* s_2^* *to* $\perp$ *contains two "*$-$*" arcs.*

The plan is now to refine the sketch shown in Figure 1 so as to specialize the application of the After and Optimize operators for the $\mathcal{L}_s$ language. This will require the introduction of further operators and procedures, which are sketchily described in Figure 3. In particular, the overall procedure receives a schema S and an update U and first transforms the schema in ato new schema S', whose integrity constraints hold in the state before the update if and only if the original constraints in S hold in the updated state. Then, the intensional database in S' is completely eliminated (through "unfolding"), because its definitions are going to be embedded in the integrity constraints themselves, thereby producing a set of integrity constraints Γ. Finally, the Optimize operator tries to remove all redundancies from Γ by eliminating literals as well as whole denials as much as possible. This is done by exploiting the consistency assumption, i.e., the fact that the original integrity constraints in S hold before the update; to do this, Optimize receives as input the integrity constraints in S after embedding the definitions of its intensional database in the constraints (again, through unfolding), which are represented by the set Δ in the sketch. The overall output is the set of integrity constraints Γ^* from which the redundancies are removed.

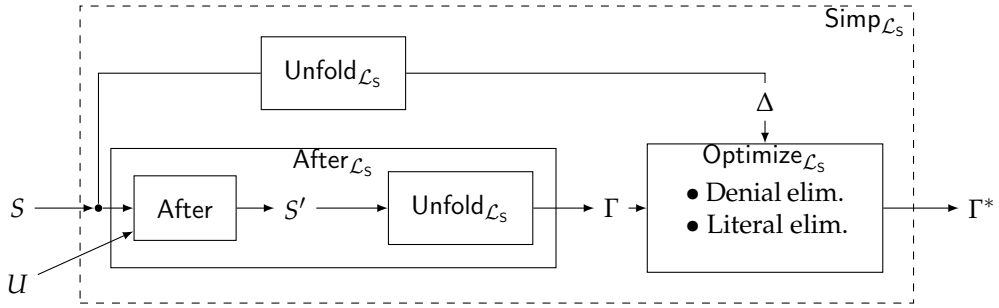

Figure 3. Sketch of the main blocks in the simplification procedure Simp$_{\mathcal{L}_s}$.

Generating Weakest Preconditions in $\mathcal{L}_s$

The following syntactic transformation, After, generates a WP.

Definition 7 (After). *Let $S = \langle IDB, \Gamma \rangle$ be a schema and U an update $p(\overrightarrow{X}) \Leftarrow p^U(\overrightarrow{X})$.*

- *Let us indicate with Γ^U a copy of Γ in which any atom $p(\overrightarrow{t})$ is simultaneously replaced by the expression $p^U(\overrightarrow{t})$ and every intensional predicate q is simultaneously replaced by a new intensional predicate q^U defined in IDB^U below.*
- *Similarly, let us indicate with IDB^U a copy of IDB in which the same replacements are simultaneously made, and let IDB^* be the biggest subset of $IDB \cup IDB^U$ including only definitions of predicates on which Γ^U depends.*

We define $\mathsf{After}^U(S) = \langle IDB^*, \Gamma^U \rangle$.

Example 5. *Consider the updates and IDB definitions of Example 1. Let schema S_1 be $\langle IDB_1, \Gamma_1 \rangle$, where $\Gamma_1 = \{\leftarrow p(X) \wedge q(X)\}$ states that p and q are mutually exclusive. We have* $\mathsf{After}^{U_1}(S_1) = \langle IDB_1, \Gamma_1^{U_1} \rangle$, *where* $\Gamma_1^{U_1} = \{\leftarrow p'(X) \wedge q(X)\}$. *We replace $p'(X)$ by its defining formula and thus omit the intensional database:*

$$\mathsf{After}^{U_1}(\Gamma_1) = \Sigma = \quad \{ \quad \leftarrow \quad p(X) \wedge q(X), \qquad \leftarrow \quad X = a \wedge q(X) \quad \}.$$

Clearly, $\mathsf{After}^U(S)$ is a WP of S with respect to U, and, trivially, a CWP.

For $\mathcal{L}_s$, we use *unfolding* [85] to replace every intensional predicate by its definition until only extensional predicates appear. To this end, we use the Unfold$_{\mathcal{L}_s}$ operator below.

Definition 8 (Unfolding). *Let $S = \langle IDB, \Gamma \rangle$ be a database schema in $\mathcal{L}_s$. Then,* $\mathsf{Unfold}_{\mathcal{L}_s}(S)$ *is the schema $\langle \emptyset, \Gamma' \rangle$, where Γ' is the set of denials obtained by iterating the two following steps as long as possible:*

1. *replace, in Γ, each atom $p(\overrightarrow{t})$ by $F^p\{\overrightarrow{X} / \overrightarrow{t}\}$, where F^p is p's defining formula and $\overrightarrow{X}$ its head variables. If no replacement was made, then stop;*
2. *transform the result into a set of denials according to the following patterns:*
   - *$\leftarrow A \wedge (B_1 \vee B_2)$ is replaced by $\leftarrow A \wedge B_1$ and $\leftarrow A \wedge B_2$;*
   - *$\leftarrow A \wedge \neg(B_1 \vee B_2)$ is replaced by $\leftarrow A \wedge \neg B_1 \wedge \neg B_2$;*
   - *$\leftarrow A \wedge \neg(B_1 \wedge B_2)$ is replaced by $\leftarrow A \wedge \neg B_1$ and $\leftarrow A \wedge \neg B_2$.*

Due to the implicit outermost universal quantification of the variables, non-distinguished variables in a predicate definition are existentially quantified to the right-hand side of the arrow, as shown in Example 6 below. For this reason, with no indication of the quantifiers, the replacements in Definition 8 preserve equivalence if no predicate containing non-distinguished variables occurs negated in the resulting expression.

Example 6. *Consider S_1 from Example 3, which, as shown, is not in $\mathcal{L}_s$. With the explicit indication of the quantifiers, we have $IDB_1 \equiv \{\forall X(s_1(X) \leftarrow \exists Y\ r_1(X,Y))\}$. The replacement, in Γ_1, of $s_1(X)$ by its definition in IDB_1 would determine the formula $\Gamma_1' = \{\forall X, Y(\leftarrow p_1(X) \wedge \neg r_1(X,Y))\}$. However, this replacement is not equivalence-preserving, because a predicate (r_1) containing a non-distinguished variable occurs negated: $\Gamma_1 \equiv \{\forall X(\leftarrow p_1(X) \wedge \neg\exists Y\ r_1(X,Y))\} \not\equiv \Gamma_1'$.*

We shall cover the cases in which $\neg\exists$ may occur in denials in Section 5.

The language $\mathcal{L}_s$ is closed under unfolding and $\mathsf{Unfold}_{\mathcal{L}_s}$ preserves equivalence. We then refine After for $\mathcal{L}_s$ as $\mathsf{After}^U_{\mathcal{L}_s}(S) = \mathsf{Unfold}_{\mathcal{L}_s}(\mathsf{After}^U(S))$.

Example 7. *Consider again a database containing information about books, where the binary predicate b contains the ISBN (first argument) and the title (second argument). We expect, for this database, updates of the form $U = \{b(X,Y) \Leftarrow b^U(X,Y)\}$, where b^U is a query defined by the predicate definition $b^U(X,Y) \leftarrow b(X,Y) \vee (X = \mathbf{i} \wedge Y = \mathbf{t})$, i.e., U is the addition of the tuple $\langle \mathbf{i}, \mathbf{t} \rangle$ to b. The following integrity constraint is given:*

$$\phi = \leftarrow b(X,Y) \wedge b(X,Z) \wedge Y \neq Z$$

meaning that no ISBN can be associated with two different titles. First, each occurrence of b is replaced by b^U, obtaining

$$\leftarrow b^U(X,Y) \wedge b^U(X,Z) \wedge Y \neq Z.$$

Then, $\mathsf{Unfold}_{\mathcal{L}_s}$ *is applied to this integrity constraint. The first step of Definition 8 generates the following:*

$$\{\leftarrow (b(X,Y) \vee (X = \mathbf{i} \wedge Y = \mathbf{t})) \wedge (b(X,Z) \vee (X = \mathbf{i} \wedge Z = \mathbf{t})) \wedge Y \neq Z\}.$$

The second step translates it to clausal form:

$$\begin{aligned} \mathsf{After}^U_{\mathcal{L}_s}(\{\phi\}) = \{ \quad & \leftarrow b(X,Y) \wedge b(X,Z) \wedge Y \neq Z, \\ & \leftarrow b(X,Y) \wedge X = \mathbf{i} \wedge Z = \mathbf{t} \wedge Y \neq Z, \\ & \leftarrow X = \mathbf{i} \wedge Y = \mathbf{t} \wedge b(X,Z) \wedge Y \neq Z, \\ & \leftarrow X = \mathbf{i} \wedge Y = \mathbf{t} \wedge X = \mathbf{i} \wedge Z = \mathbf{t} \wedge Y \neq Z \quad \}. \end{aligned}$$

4.2. Simplification in $\mathcal{L}_s$

The result returned by $\mathsf{After}_{\mathcal{L}_s}$ may contain redundant parts (e.g., $a = a$) and does not exploit the consistency assumption. For this purpose, we define a transformation $\mathsf{Optimize}_{\mathcal{L}_s}$ that optimizes a given constraint theory using a set of trusted hypotheses. We describe here an implementation in terms of sound and terminating rewrite rules. Among the tools we use are subsumption, reduction, and resolution.

Definition 9 (Subsumption). *Given two denials ϕ_1 and ϕ_2, ϕ_1* subsumes *ϕ_2 (via σ), written $\phi_1 \sqsubseteq \phi_2$, if there is a substitution σ such that each literal in $\phi_1\sigma$ occurs in ϕ_2. The subsumption is* strict, *written $\phi_1 \sqsubset \phi_2$, if ϕ_1 is not a variant of ϕ_2.*

The subsumption algorithm (see, e.g., [86]), besides checking subsumption, also returns the substitution σ. As is well known, the subsuming denial implies the subsumed one.

Example 8. *$\leftarrow p(X, Y) \wedge q(Y)$ subsumes $\leftarrow p(X, b) \wedge X \neq a \wedge q(b)$ via $\{Y/b\}$.*

Reduction [87] characterizes the elimination of redundancies within a single denial.

Definition 10 (Reduction). *For a denial ϕ, the reduction ϕ^- of ϕ is the result of applying the following rules on ϕ as long as possible, where L is a literal, c_1, c_2 are distinct constants, X is a bound variable, t is a term, A is an atom, C, D (possibly empty) are conjunctions of literals, and* vars *indicates the set of bound variables occurring in its argument.*

$$
\begin{array}{rcl}
\leftarrow L \wedge C & \Rightarrow & \leftarrow C \quad \textit{if } L \textit{ is of the form } t = t \textit{ or } c_1 \neq c_2 \\
\leftarrow L \wedge C & \Rightarrow & \textit{true} \quad \textit{if } L \textit{ is of the form } t \neq t \textit{ or } c_1 = c_2 \\
\leftarrow X = t \wedge C & \Rightarrow & \leftarrow C\{X/t\} \\
\leftarrow A \wedge \neg A \wedge C & \Rightarrow & \textit{true} \\
\leftarrow C \wedge D & \Rightarrow & \leftarrow D \quad \textit{if } (\leftarrow C) \sqsubseteq (\leftarrow D) \textit{ with a substitution } \sigma \textit{ s.t.} \\
 & & \mathrm{dom}(\sigma) \cap \mathrm{vars}(D) = \emptyset
\end{array}
$$

Clearly, $\phi^- \equiv \phi$. The last rewrite rule is called subsumption factoring [88] and includes the elimination of duplicate literals from a denial as a special case. An additional rule for handling parameters may be considered in the reduction process:

$$\leftarrow \mathbf{a} = c_1 \wedge C \quad \Rightarrow \quad \leftarrow \mathbf{a} = c_1 \wedge C\{\mathbf{a}/c_1\} \tag{1}$$

This may replace parameters with constants and possibly allow further reduction. For example, the denial $\leftarrow \mathbf{a} = c \wedge b = \mathbf{a}$ would be transformed into $\leftarrow \mathbf{a} = c \wedge b = c$, and, thus, into *true*, since b and c are different constants.

We now briefly recall the definition of resolvent and derivation for the well-known principle of *resolution* [89,90] and cast it to the context of deductive databases.

Definition 11. *Let $\phi_1' = \leftarrow L_1 \wedge \ldots \wedge L_m, \phi_2' = \leftarrow M_1 \wedge \ldots \wedge M_n$ be two standardized apart variants of denials ϕ_1, ϕ_2. If θ is a mgu of $\{L_i, \neg M_j\}$, then the clause*

$$\leftarrow (L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots L_m \wedge M_1 \wedge \ldots \wedge M_{j-1} \wedge M_{j+1} \wedge \ldots \wedge M_n)\theta$$

is a binary resolvent *of ϕ_1 and ϕ_2 and L_i, M_j are said to be the literals* resolved upon.

The resolution principle is a sound inference rule, in that a resolvent is a logical consequence of its parent clauses, and preserves range restriction [91].

We also refer to the notion of *expansion* [8]: the expansion of a clause consists of replacing every constant or parameter in a database predicate (or variable already appearing elsewhere in database predicates) by a new variable and adding the equality between the new variable and the replaced item. We indicate the expansion of a (set of) denial(s) with a "+" superscript.

Example 9. *Let* $\phi = \leftarrow p(X, a, X)$. *Then,* $\phi^+ = \leftarrow p(X, Y, Z) \wedge Y = a \wedge Z = X$.

For a constraint theory Γ in $\mathcal{L}_s$ and a denial ϕ in $\mathcal{L}_s$, the notation $\Gamma \vdash_R \phi$ indicates that there is a resolution derivation of a denial ψ from Γ^+ and $\psi^- \sqsubseteq \phi$. The $\vdash_R$ procedure is sound; additionally, it is guaranteed to terminate on any input provided that, in each resolution step, the resolvent has at most as many literals as those in the largest denial in Γ^+ (i.e., by forcing an upper bound on the size of the clauses).

We can now provide a possible implementation of an optimization operator that eliminates redundant literals and denials from a given constraint theory Γ assuming that another theory Δ holds. Informally, $\vdash_R$, subsumption and reduction are used to approximate entailment. In the following, $A \sqcup B$ indicates the union of disjoint sets.

Definition 12. *Given two constraint theories* Δ *and* Γ *in* $\mathcal{L}_s$, $\mathsf{Optimize}^{\Delta}_{\mathcal{L}_s}(\Gamma)$ *is the result of applying the following rewrite rules on* Γ *as long as possible. In the following,* ϕ, ψ *are denials in* $\mathcal{L}_s$, Γ' *is a constraint theory in* $\mathcal{L}_s$.

$$\begin{array}{lcl}
\{\phi\} \sqcup \Gamma' & \Rightarrow & \Gamma' \textit{ if } \phi^- = \textit{true} \\
\{\phi\} \sqcup \Gamma' & \Rightarrow & \Gamma' \textit{ if } (\Gamma' \cup \Delta) \vdash_R \phi \\
\{\phi\} \sqcup \Gamma' & \Rightarrow & \{\phi^-\} \cup \Gamma' \textit{ if } \phi \neq \phi^- \neq \textit{true} \\
\{\phi\} \sqcup \Gamma' & \Rightarrow & \{\psi^-\} \cup \Gamma' \textit{ if } (\{\phi\} \sqcup \Gamma' \cup \Delta) \vdash_R \psi \textit{ and } \psi^- \sqsubset \phi
\end{array}$$

The first two rules attempt the elimination of a whole denial, whereas the last two try to remove literals from a denial. We can now fully define simplification for $\mathcal{L}_s$:

Definition 13. *Let* $S = \langle IDB, \Gamma \rangle \in \mathcal{L}_s$ *and* U *be an update in* $\mathcal{L}_s$ *with respect to* S. *Let* $\mathsf{Unfold}_{\mathcal{L}_s}(S) = \langle \emptyset, \Gamma' \rangle$. *We define* $\mathsf{Simp}^U_{\mathcal{L}_s}(S) = \mathsf{Optimize}^{\Gamma'}_{\mathcal{L}_s}(\mathsf{After}^U_{\mathcal{L}_s}(S))$.

Clearly, $\mathsf{Simp}^U_{\mathcal{L}_s}(S)$ is a CWP of S with respect to U.

Example 10. *Consider again Example 7. The reduction of each denial in* $\mathsf{After}^U_{\mathcal{L}_s}(\{\phi\})$ *generates the following set.*

$$\begin{array}{lll}
\{ & \leftarrow & b(X, Y) \wedge b(X, Z) \wedge Y \neq Z, \\
 & \leftarrow & b(\mathbf{i}, Y) \wedge Y \neq \mathbf{t}, \\
 & \leftarrow & b(\mathbf{i}, Z) \wedge \mathbf{t} \neq Z \qquad\qquad \}.
\end{array}$$

Then, the third denial is removed, as it is subsumed by the second one; the first constraint is subsumed by ϕ *and, thus, removed, so* $\mathsf{Simp}^U_{\mathcal{L}_s}(\{\phi\}) = \{\leftarrow b(\mathbf{i}, Y) \wedge Y \neq \mathbf{t}\}$. *This result indicates that, for the database to be consistent after update* U, *a book with ISBN* $\mathbf{i}$ *must not be already associated with a title* Y *different from* $\mathbf{t}$.

5. Denials with Negated Existential Quantifiers

In the definition of $\mathcal{L}_s$, we limited the level of interaction between negation and existential quantification in the constraint theories. We now relax this limitation and extend the syntax of denials so as to allow the presence of negated existential quantifiers. The simplification procedure can be adapted to such cases, provided that its components are adjusted so as to handle conjuncts starting with a negated existential quantifier.

In this section, we address the problem of simplification of integrity constraints in *hierarchical* databases, in which all clauses are range-restricted and the schemata are non-recursive, but there is no other restriction on the occurrence of negation. We refer to the language of such schemata as $\mathcal{L}_{\text{H}}$.

Definition 14 ($\mathcal{L}_{\text{H}}$)**.** *Let S be a schema. S is in $\mathcal{L}_{\text{H}}$ if its starred dependency graph is acyclic.*

Similarly to the previous section, we summarize the notation and operators used in this section in tabular form (see Table 2), which add to those summarized in Table 1. A diagram showing the main blocks composing the simplification procedure in $\mathcal{L}_{\text{H}}$ would look exactly like the one shown in Figure 3, after taking care of replacing each occurrence of $\mathcal{L}_{\text{S}}$ with $\mathcal{L}_{\text{H}}$.

Unfolding predicates in integrity constraints with respect to their definitions cannot be done in the same way as $\mathsf{Unfold}_{\mathcal{L}_{\text{S}}}$. For this purpose, we extend the syntax of denials so as to allow negated existential quantifiers to occur in literals.

Table 2. Operators used for the simplification procedure, as defined in Section 5.

Notation	Explanation
$\mathcal{L}_{\text{H}}$	A class of schemata that can be unfolded as a set of extended denials (Definition 14)
$\Gamma = \mathsf{Unfold}_{\mathcal{L}_{\text{H}}}(S)$	For a schema $S = \langle IDB, IC \rangle \in \mathcal{L}_{\text{H}}$, Γ is the unfolding of IC wrt IDB (Definition 16)
$\Gamma = \mathsf{After}^{U}_{\mathcal{L}_{\text{H}}}(S)$	The set of extended denials Γ is a WP of schema $S \in \mathcal{L}_{\text{H}}$ wrt update U (Definition 17)
$\phi_1 \hat{\sqsubseteq} \phi_2$	Extended denial ϕ_1 extended-subsumes extended denial ϕ_2 (Definition 19)
ϕ^{-}	Reduction: same as extended denial ϕ but without redundancies (Definition 20)
$\Gamma' = \mathsf{Optimize}^{\Delta}_{\mathcal{L}_{\text{H}}}(\Gamma)$	Γ' is the result of applying (possibly nested) extended denial elimination and general literal elimination from constraint theory $\Gamma \in \mathcal{L}_{\text{H}}$ as long as possible by trusting constraint theory $\Delta \in \mathcal{L}_{\text{H}}$ (Definition 21)
$\Phi = \mathsf{Simp}^{U}_{\mathcal{L}_{\text{H}}}(S)$	The set of extended denials Φ is a CWP of schema $S \in \mathcal{L}_{\text{H}}$ wrt update U computed through $\mathsf{Optimize}_{\mathcal{L}_{\text{H}}}$ and $\mathsf{After}_{\mathcal{L}_{\text{H}}}$ (Definition 22)

Definition 15 (Extended denials)**.** *A* negated existential expression *or* NEE *is an expression of the form $\neg\exists \overrightarrow{X}\, B$, where B is called the* body *of the NEE, $\overrightarrow{X}$ are some (possibly all) of the variables occurring in B, and B has the form $L_1 \wedge \ldots \wedge L_n$, where each L_i is a* general literal*, i.e., either a literal or an NEE.*

A formula of the form $\forall \overrightarrow{X}\, (\leftarrow B)$, where B is the body of an NEE and $\overrightarrow{X}$ are some (possibly all) of the free variables in B, is called an extended denial*. When there is no ambiguity on the variables in $\overrightarrow{X}$, extended denials are simply written $\leftarrow B$.*

Example 11. *The formula $\leftarrow parent(X) \wedge \neg\exists Y child_of(X, Y)$ is an extended denial. It reads as follows: there is inconsistency if there is a parent X that does not have a child. Note that this is different from the (non-range-restricted) denial $\leftarrow parent(X) \wedge \neg child_of(X, Y)$, which states that if X is a parent then all individuals must be his/her children.*

We observe that variables under a negated existential quantifier conform with the intuition behind safeness, so we could conclude that the first formula in Example 11 is safe, whereas the second one is not.

The framework we have defined so far is very expressive, since it captures the most common and useful kinds of dependencies, such as TGDs and EGDs. A TGD is a formula of the form $\forall \overrightarrow{X} (\phi(\overrightarrow{X}) \rightarrow \exists \overrightarrow{Y}\, \psi(\overrightarrow{X}, \overrightarrow{Y}))$, where $\phi(\overrightarrow{X})$ is a conjunction of atomic formulas, all with variables among the variables in $\overrightarrow{X}$; every variable in $\overrightarrow{X}$ appears in $\phi(\overrightarrow{X})$ (but

not necessarily in ψ), and $\psi(\overrightarrow{X}, \overrightarrow{Y})$ is a conjunction of atoms, all with variables among $\overrightarrow{X}$ and $\overrightarrow{Y}$. Clearly, such a TGD can be expressed with our notation as the extended denial $\leftarrow \phi(\overrightarrow{X}) \wedge \neg\exists \overrightarrow{Y}\, \psi(\overrightarrow{X}, \overrightarrow{Y})$. EGDs are formulas of the form $\forall \overrightarrow{X}\, (\phi(\overrightarrow{X}) \rightarrow X_1 = X_2)$, where X_1 and X_2 are variables in $\overrightarrow{X}$. Clearly, this is expressed as a denial $\leftarrow \phi(\overrightarrow{X}) \wedge X_1 \neq X_2$.

We can now apply unfolding in $\mathcal{L}_{\text{H}}$ to obtain extended denials. In doing so, attention needs to be paid when replacing negated intensional predicates by their definition, since they may contain non-distinguished variables, meaning that existential quantifiers have to be explicitly indicated. As was the case for $\text{Unfold}_{\mathcal{L}_{\text{S}}}$, the replacements may result in disjunctions and negated conjunctions. Therefore, additional steps are needed to restore the extended denial form.

Definition 16. *Let $S = \langle IDB, \Gamma \rangle$ be a database schema in $\mathcal{L}_{\text{H}}$. We define $\text{Unfold}_{\mathcal{L}_{\text{H}}}(S)$ as the set of extended denials obtained by iterating the two following steps as long as possible:*

1. *replace, in Γ, each occurrence of a literal of the form $\neg p(\overrightarrow{t})$ by $\neg\exists \overrightarrow{Y}\, F^p\{\overrightarrow{X} / \overrightarrow{t}\}$ and of a literal of the form $p(\overrightarrow{t})$ by $F^p\{\overrightarrow{X} / \overrightarrow{t}\}$, where F^p is p's defining formula, $\overrightarrow{X}$ its head variables and $\overrightarrow{Y}$ its non-distinguished variables. If no replacement was made, stop;*
2. *transform the resulting formula into a set of extended denials according to the following patterns; $\Phi(Arg)$ is an expression indicating the body of an NEE in which Arg occurs; $\overrightarrow{X}$ and $\overrightarrow{Y}$ are disjoint sequences of variables:*
   - *$\leftarrow A \wedge (B \vee C)$ is replaced by $\leftarrow A \wedge B$ and $\leftarrow A \wedge C$;*
   - *$\leftarrow A \wedge \neg(B \vee C)$ is replaced by $\leftarrow A \wedge \neg B \wedge \neg C$;*
   - *$\leftarrow A \wedge \neg(B \wedge C)$ is replaced by $\leftarrow A \wedge \neg B$ and $\leftarrow A \wedge \neg C$;*
   - *$\leftarrow A \wedge \neg\exists \overrightarrow{X} \Phi(\neg\exists \overrightarrow{Y}\, [B \wedge (C \vee D)])$ is replaced by $\leftarrow A \wedge \neg\exists \overrightarrow{X} \Phi(\neg\exists \overrightarrow{Y}\, [B \wedge C] \wedge \neg\exists \overrightarrow{Y}\, [B \wedge D])$.*

Without loss of generality, we can assume that, for any NEE $N = \neg\exists \overrightarrow{X}\, B$ occurring in an extended denial ϕ, the variables $\overrightarrow{X}$ do not occur outside N in ϕ. This can simply be obtained by renaming the variables appropriately and we refer to such an extended denial as *standardized*. The *level* of an NEE in an extended denial is the number of NEEs that contain it plus 1. The level of a variable X in a standardized extended denial is the level of the NEE starting with $\neg\exists \overrightarrow{X}$, where X is one of the variables in $\overrightarrow{X}$, or 0 if there is no such NEE. The level of an extended denial is the maximum level of its NEEs, or 0 if there is no NEE. In Example 11, the extended denial has level 1, X has level 0, and Y has level 1.

With a slight abuse of notation, we write in the following $S \equiv \Psi$ (or $\Psi \equiv S$), where $S = \langle IDB, IC \rangle$ is a schema and Ψ is a set of extended denials, to indicate that, for every database D based on IDB, $D \models IC$ if $D \models \Psi$. We can now claim the correctness of $\text{Unfold}_{\mathcal{L}_{\text{H}}}$. We state the following proposition without a proof, since all steps in Definition 16 are trivially equivalence-preserving.

Proposition 1. *Let $S \in \mathcal{L}_{\text{H}}$. Then $\text{Unfold}_{\mathcal{L}_{\text{H}}}(S) \equiv S$.*

Since the variables under a negated existential quantifier conform with the intuition behind safeness, the unfolding of a schema in which all the clauses are range-restricted yields a set of extended denials that still are safe in this sense. We also note that the language of extended denials is very expressive. In [92], it was shown that *any* closed formula of the form $\forall \overrightarrow{X}\, (\leftarrow B)$, where B is a first-order formula and $\overrightarrow{X}$ are its open variables, can be equivalently expressed by a set of Prolog rules plus the denial $\leftarrow q(\overrightarrow{X})$, where q is a fresh predicate symbol. The construction is such that, if B is function-free, then the resulting rule set is also function-free (no skolemization is needed), i.e., it is that of a schema in $\mathcal{L}_{\text{H}}$. The unfolding of the obtained schema is therefore an equivalent set of extended denials.

A simplification procedure can now be constructed for extended denials in a way similar to what was achieved in $\mathcal{L}_S$.

Definition 17. *Let S be a schema in $\mathcal{L}_H$ and U an update. We define*

$$\mathsf{After}^U_{\mathcal{L}_H}(S) = \mathsf{Unfold}_{\mathcal{L}_H}(\mathsf{After}^U(S)).$$

The optimization step needs to take into account the nesting of NEEs in extended denials. Besides the elimination of disjunctions within NEEs, which is performed by $\mathsf{Unfold}_{\mathcal{L}_H}$, we can also eliminate, from an NEE, equalities and non-equalities referring to variables of lower level with respect to the NEE.

Definition 18. *Let A, B, C be (possibly empty) conjunctions of general literals, $\overrightarrow{Y}$, $\overrightarrow{Z}$ disjoint (sequences of) variables, W a variable of level lower than the level of $\overrightarrow{Z}$, and $\Phi(Arg)$ an expression indicating an NEE in which Arg occurs. The following rewrite rules are, respectively, the* equality elimination *and* non-equality elimination *rules.*

$$\begin{array}{l} \leftarrow A \wedge \Phi(\neg\exists \overrightarrow{Y}[B \wedge \neg\exists \overrightarrow{Z}(C \wedge W = c)]) \;\Rightarrow \\ \leftarrow A \wedge \Phi(\neg\exists \overrightarrow{Y}[B \wedge W = c \wedge \neg\exists \overrightarrow{Z}(C)] \wedge \neg\exists \overrightarrow{Y}[B \wedge W \neq c]) \end{array}$$

$$\begin{array}{l} \leftarrow A \wedge \Phi(\neg\exists \overrightarrow{Y}[B \wedge \neg\exists \overrightarrow{Z}(C \wedge W \neq c)]) \;\Rightarrow \\ \leftarrow A \wedge \Phi(\neg\exists \overrightarrow{Y}[B \wedge W \neq c \wedge \neg\exists \overrightarrow{Z}(C)] \wedge \neg\exists \overrightarrow{Y}[B \wedge W = c]) \end{array}$$

The above (non-)equality elimination rewrite rules are equivalence-preserving, as stated below.

Proposition 2. *Let ψ be an extended denial and ψ' (resp. ψ'') be the extended denial obtained after an application of the equality (resp. non-equality) elimination rule. Then $\psi \equiv \psi'$ and $\psi \equiv \psi''$.*

Proof. Using the notation of Definition 18, we have the following:

$$\begin{array}{rl} & \neg\exists \overrightarrow{Y}[B \wedge \neg\exists \overrightarrow{Z}(C \wedge W = c)] \\ \equiv & \neg\exists \overrightarrow{Y}[B \wedge (W = c \vee W \neq c) \wedge \neg\exists \overrightarrow{Z}(C \wedge W = c)] \\ \equiv & \neg\exists \overrightarrow{Y}[B \wedge W = c \wedge \neg\exists \overrightarrow{Z}(C \wedge W = c)] \wedge \\ & \neg\exists \overrightarrow{Y}[B \wedge W \neq c \wedge \neg\exists \overrightarrow{Z}(C \wedge W = c)] \\ \equiv & \neg\exists \overrightarrow{Y}[B \wedge W = c \wedge \neg\exists \overrightarrow{Z}(C)] \wedge \neg\exists \overrightarrow{Y}[B \wedge W \neq c] \end{array}$$

In the first step, we added the tautological conjunct $(W = c \vee W \neq c)$. In the second step, we used de Morgan's laws in order to eliminate the disjunction (as in the definition of $\mathsf{Unfold}_{\mathcal{L}_H}$). The formula $W = c \wedge \neg\exists \overrightarrow{Z}(C \wedge W = c)$ can be rewritten as $W = c \wedge \neg(\exists \overrightarrow{Z} C \wedge W = c)$, and then as $W = c \wedge (\neg\exists \overrightarrow{Z}(C) \vee W \neq c)$, which, with a resolution step, results in the first NEE in the last extended denial. Similarly, $W \neq c \wedge \neg\exists \overrightarrow{Z}(C \wedge W = c)$ can be rewritten as $W \neq c \wedge (\neg\exists \overrightarrow{Z}(C) \vee W \neq c)$, which results (by absorption) in the second NEE in the last extended denial.

The proof is similar for non-equality elimination. □

In cases where only levels 0 and 1 are involved, the rules look simpler. For example, equality elimination can be conveniently formulated as follows.

$$\leftarrow B \wedge \neg\exists \overrightarrow{X}[C \wedge W = c] \quad \Rightarrow \quad \left\{ \begin{array}{l} \leftarrow B\{W/c\} \wedge \neg\exists \overrightarrow{X} C\{W/c\} \\ \leftarrow B \wedge W \neq c \end{array} \right\}. \tag{2}$$

Although (non-)equality elimination does not necessarily shorten the input formula (in fact, it can also lengthen it), it always reduces the number of literals in higher-level NEEs. Therefore, convergence to termination can still be guaranteed if this rewrite rule is applied during optimization. Repeated application of such rules "pushes" outwards the involved (non-)equalities until they reach an NEE whose level is the same as the level of the variable in the (non-)equality. Then, in case of an equality, the usual equality elimination step of reduction can be applied.

Example 12. *The following rewrites show the propagation of a variable of level 0 (X) from level* 2 *to level* 0 *via two equality eliminations and one non-equality elimination.*

$$
\begin{array}{lll}
 & \leftarrow & p(X) \wedge \neg\exists Y\{q(X,Y) \wedge \neg\exists Z[r(X,Y,Z) \wedge X = a]\} \\
\equiv & \leftarrow & p(X) \wedge \neg\exists Y\{q(X,Y) \wedge X = a \wedge \neg\exists Z[r(X,Y,Z)]\} \\
 & & \wedge\neg\exists Y[q(X,Y) \wedge X \neq a] \\
\equiv \{ & \leftarrow & p(a) \wedge \neg\exists Y[q(a,Y) \wedge \neg\exists Z r(a,Y,Z)] \wedge \neg\exists Y[q(a,Y) \wedge a \neq a], \\
 & \leftarrow & p(X) \wedge X \neq a \wedge \neg\exists Y[X \neq a \wedge q(X,Y)] \quad \} \\
\equiv \{ & \leftarrow & p(a) \wedge \neg\exists Y[q(a,Y) \wedge \neg\exists Z r(a,Y,Z)] \wedge \neg\exists Y[q(a,Y) \wedge a \neq a], \\
 & \leftarrow & p(X) \wedge X \neq a \wedge \neg\exists Y q(X,Y), \\
 & \leftarrow & p(a) \wedge X \neq a \wedge X = a \quad \} \\
\equiv \{ & \leftarrow & p(a) \wedge \neg\exists Y[q(a,Y) \wedge \neg\exists Z r(a,Y,Z)], \\
 & \leftarrow & p(X) \wedge X \neq a \wedge \neg\exists Y q(X,Y) \quad \}
\end{array}
$$

In the first step, we applied equality elimination to $X = a$ *at level* 2*. In the second step, we applied the rewrite rule* (2) *for equality elimination to* $X = a$ *at level 1. Then, we applied non-equality elimination to* $X \neq a$ *at level* 1 *in the second denial. In the last step, we removed, by standard application of reduction, the last extended denial and the last NEE in the first extended denial, which are clearly tautological.*

We observe that the body of an NEE is structurally similar to the body of an extended denial. The only difference is that, in the former, there are variables that are quantified at a lower level. According to this observation, such (free) variables in the body of an NEE are to be treated as parameters during the different optimization steps, since, as was indicated on page 5, parameters are free variables.

Reduction (Definition 10) can then take place in NEE bodies exactly as in ordinary denials, with the proviso above of treating free variables as parameters.

The definition of resolution (Definition 11) can be adapted for extended denials by applying it to general literals instead of literals.

Subsumption (Definition 9) can also be applied to extended denials without changing the definition. However, we can slightly modify the notion of subsumption to explore the different levels of NEEs in an extended denial. This is captured by the following definition.

Definition 19. *Let* $\phi = \leftarrow A \wedge B$ *and* $\psi = \leftarrow C \wedge D$ *be two extended denials, where A and C are (possibly empty) conjunctions of literals and B and D are (possibly empty) conjunctions of NEEs. Then* ϕ *extended-subsumes* ψ*, written* $\phi \hat{\subseteq} \psi$*, if both conditions (1) and (2) below hold.*

(1) $\leftarrow A$ *subsumes* $\leftarrow C$ *with substitution* σ*.*

(2) *For every NEE* $\neg\exists \overrightarrow{X}_N N$ *in* B, *there is an NEE* $\neg\exists \overrightarrow{X}_M M$ *in* D *such that* $\leftarrow N\sigma$ *is extended-subsumed by* $\leftarrow M$.

Example 13. *The extended denial* $\leftarrow p(X) \wedge \neg\exists Y, Z[q(X,Y) \wedge r(Y,Z)]$ *extended-subsumes the extended denial* $\leftarrow p(a) \wedge \neg\exists T[q(a,T)] \wedge \neg\exists W[s(T)]$, *since* $\leftarrow p(X)$ *subsumes* $\leftarrow p(a)$ *with substitution* $\{X/a\}$ *and, in turn,* $\leftarrow q(a,T)$ *subsumes* $\leftarrow q(a,Y) \wedge r(Y,Z)$.

This definition encompasses ordinary subsumption, in that it coincides with it if B and D are empty. Furthermore, it captures the desired property that if ϕ extended-subsumes ψ, then $\phi \models \psi$; the reverse, as in subsumption, does not necessarily hold.

Proposition 3. *Let* ϕ *and* ψ *be extended denials. If* $\phi \hat{\sqsubseteq} \psi$, *then* $\phi \models \psi$.

Proof. Let ϕ, ψ, A, B, C, D be as in Definition 19. If B and D are empty, the claim holds, since ϕ and ψ are ordinary denials. The claim also holds if D is not empty, since $\leftarrow C$ entails $\leftarrow C \wedge D$. We now show the general claim with an inductive proof on the level of extended denials.

The base case (level 0) is already proven.

Inductive step. Suppose now that ϕ is of level $n+1$ and that the claim holds for extended denials of level n or less. Assume as a first case that B is empty. Then, ϕ entails ψ, since ϕ entails $\leftarrow C$ ($\leftarrow A$ subsumes $\leftarrow C$ by hypothesis). Assume for the moment that $B = \neg\exists \overrightarrow{X}_N N$ is an NEE of level 1 in ϕ and that D contains an NEE (of level 1) $\neg\exists \overrightarrow{X}_M M$, such that $\phi' = \leftarrow N\sigma$ is extended-subsumed by $\psi' = \leftarrow M$, as assumed in the hypotheses. However, ϕ' and ψ' are extended denials of level n and, therefore, if ψ' subsumes ϕ', then ψ' entails ϕ' by inductive hypothesis. Clearly, since $\leftarrow A$ entails $\leftarrow C \wedge D$ (by hypothesis) and $\leftarrow M$ entails $\leftarrow N\sigma$ (as a consequence of the inductive hypothesis), then $\leftarrow A \wedge B$ entails $\leftarrow C \wedge D$, which is our claim. If B contains more than one NEE of level 1, the argument is iterated by adding one NEE at a time. □

The inductive proof also shows how to check extended subsumption with a finite number of subsumption tests. This implies that extended subsumption is decidable, since subsumption is. Now that a correct extended subsumption is introduced, it can be used instead of subsumption in the subsumption factoring rule of reduction (Definition 10). In the following, when referring to an NEE $N = \neg\exists \overrightarrow{X} B$, we can also write it as a denial $\leftarrow B$, with the understanding that the free variables in N are considered parameters.

Definition 20. *For an extended denial* ϕ, *the reduction* ϕ^- *of* ϕ *is the result of applying on* ϕ *equality and non-equality elimination as long as possible, and then the rules of Definition 10 (reduction) on* ϕ *and its NEEs as long as possible, where "literal" is replaced by "general literal", "subsumes" by "extended-subsumes", and "denial" by "extended denial".*

Without reintroducing similar definitions, we assume that the same word replacements are made for the notion of $\vdash_R$. The underlying notions of substitution and unification also apply to extended denials and general literals; however, after substitution with a constant or parameter, the existential quantifier of a variable is removed. For example, the extended denials $\phi = \leftarrow p(X,b) \wedge \neg\exists Z[q(Z,X)]$ and $\psi = \leftarrow p(a,Y) \wedge \neg q(c,a)$ unify with substitution $\{X/a, Y/b, Z/c\}$. By virtue of the similarity between denial bodies and NEE bodies, we extend the notion of optimization as follows.

Definition 21. *Given two sets of extended denials* Δ *and* Γ, $\mathsf{Optimize}^{\Delta}_{\mathcal{L}_H}(\Gamma)$ *is the result of applying the following rewrite rules and the rules of Definition 12* ($\mathsf{Optimize}_{\mathcal{L}_S}$) *on* Γ *as long as possible.*

In the following, ϕ and ψ are NEEs, Γ' is a set of extended denials, and $\Phi(Arg)$ is an expression indicating the body of an extended denial in which Arg occurs.

$$\begin{array}{lcl}
\{\leftarrow \Phi(\phi)\} \sqcup \Gamma' & \Rightarrow & \{\leftarrow \Phi(true)\} \cup \Gamma' \text{ if } \phi^- = true \\
\{\leftarrow \Phi(\phi)\} \sqcup \Gamma' & \Rightarrow & \{\leftarrow \Phi(true)\} \cup \Gamma' \text{ if } (\Gamma' \cup \Delta) \vdash_R \phi \\
\{\leftarrow \Phi(\phi)\} \sqcup \Gamma' & \Rightarrow & \{\leftarrow \Phi(\phi^-)\} \cup \Gamma' \text{ if } \phi \neq \phi^- \neq true \\
\{\leftarrow \Phi(\phi)\} \sqcup \Gamma' & \Rightarrow & \{\leftarrow \Phi(\phi^-)\} \cup \Gamma' \text{ if } (((\{\phi\} \cup \{\leftarrow \Phi(\phi)\}) \sqcup \Gamma' \cup \Delta) \vdash_R \psi \\
 & & \text{and } \psi^- \text{ strictly extended-subsumes } \phi
\end{array}$$

Finally, the simplification procedure for $\mathcal{L}_{\text{H}}$ is composed in terms of $\mathsf{After}_{\mathcal{L}_{\text{H}}}$ and $\mathsf{Optimize}_{\mathcal{L}_{\text{H}}}$.

Definition 22. *Consider a schema $S = \langle IDB, \Gamma \rangle \in \mathcal{L}_{\text{H}}$ and an update U. Let $\Gamma' = \mathsf{Unfold}_{\mathcal{L}_{\text{H}}}(S)$. We define*

$$\mathsf{Simp}^{U}_{\mathcal{L}_{\text{H}}}(S) = \mathsf{Optimize}^{\Gamma'}_{\mathcal{L}_{\text{H}}}(\mathsf{After}^{U}_{\mathcal{L}_{\text{H}}}(S)).$$

Similarly to $\mathcal{L}_{\text{S}}$, soundness of the optimization steps and the fact that After returns a WP entail the following.

Proposition 4. *Let $S \in \mathcal{L}_{\text{H}}$ and U be an update. Then, $\mathsf{Simp}^{U}_{\mathcal{L}_{\text{H}}}(S)$ is a CWP of S with respect to U.*

Practical Applications

We now discuss the most complex non-recursive examples that we found in the literature for testing the effectiveness of the proposed simplification procedure.

Example 14. *This example is taken from [10]. Consider a schema $S = \langle IDB, \Gamma \rangle$ with three extensional predicates a, b, c, two intensional predicates p, q, a constraint theory Γ, and a set of trusted hypotheses Δ.*

$$\begin{array}{rll}
IDB = \{ & p(X,Y) \leftarrow a(X,Z) \wedge b(Z,Y), & \\
 & q(X,Y) \leftarrow p(X,Z) \wedge c(Z,Y) & \} \\
\Gamma = \{ & \leftarrow p(X,X) \wedge \neg q(1,X) & \} \\
\Delta = \{ & \leftarrow a(1,5) & \}
\end{array}$$

This schema S is not in $\mathcal{L}_{\text{S}}$ and the unfolding of S is as follows.

$$\mathsf{Unfold}_{\mathcal{L}_{\text{H}}}(S) = \{\leftarrow a(X,Y) \wedge b(Y,X) \wedge \neg\exists W, Z(a(1,W) \wedge b(W,Z) \wedge c(Z,X))\}.$$

We want to verify that the update $U = \{b(X,Y) \Leftarrow b(X,Y) \wedge X \neq 5\}$ (the deletion of all b-tuples in which the first argument is 5) does not affect consistency. $\mathsf{After}^{U}_{\mathcal{L}_{\text{H}}}(S)$ results in the following extended denial:

$$\leftarrow a(X,Y) \wedge b(Y,X) \wedge Y \neq 5 \wedge \neg\exists W, Z(a(1,W) \wedge b(W,Z) \wedge W \neq 5 \wedge c(Z,X)).$$

As previously described, during the optimization process, the last conjunct can be processed as a separate denial $\phi = \leftarrow a(1,W) \wedge b(W,Z) \wedge W \neq 5 \wedge c(Z,\mathbf{X})$, where $\mathbf{X}$ is a free variable that can be treated as a parameter (and thus indicated in bold). With a resolution step with Δ, the literal $W \neq 5$ is proved to be redundant and can thus be removed from ϕ. The obtained formula is then subsumed by $\mathsf{Unfold}_{\mathcal{L}_{\text{H}}}(S)$ and therefore $\mathsf{Simp}^{U}_{\mathcal{L}_{\text{H}}}(S) = \emptyset$, i.e., the update cannot violate the integrity constraint, which is the same result that was found in [10].

In order to simplify the notation for tuple additions and deletions, we write $p(\rightarrow a)$ as a shorthand for the database update $p(\overrightarrow{X}) \Leftarrow p(\overrightarrow{X}) \vee \overrightarrow{X} = \rightarrow a$ and $\neg p(\rightarrow a)$ for $p(\overrightarrow{X}) \Leftarrow p(\overrightarrow{X}) \wedge \overrightarrow{X} \neq \rightarrow a$.

Example 15. *The following schema S is the relevant part of an example described in [11] on page 24.*

$$\begin{array}{rrcll} S = \langle\{ & married_to(X,Y) & \leftarrow & parent(X,Z) \wedge parent(Y,Z) \wedge & \\ & & & man(X) \wedge woman(Y), & \\ & married_man(X) & \leftarrow & married_to(X,Y), & \\ & married_woman(X) & \leftarrow & married_to(Y,X), & \\ & unmarried(X) & \leftarrow & man(X) \wedge \neg married_man(X), & \\ & unmarried(X) & \leftarrow & woman(X) \wedge \neg married_woman(X) & \}, \\ \{ & & \leftarrow & man(X) \wedge woman(X), & \\ & & \leftarrow & parent(X,Y) \wedge unmarried(X) & \}\rangle \end{array}$$

If we reformulate the example using the shorthand notation, the database is updated with $U = \{man(\mathbf{a})\}$*, where* $\mathbf{a}$ *is a parameter. The unfolding given by* $\mathsf{Unfold}_{\mathcal{L}_H}(S)$ *is as follows, where m, w, p respectively, abbreviate man, woman, parent, which are the only extensional predicates.*

$$\begin{array}{ll} \{ & \leftarrow m(X) \wedge w(X), \\ & \leftarrow p(X,Y) \wedge m(X) \wedge \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge m(X) \wedge w(T)], \\ & \leftarrow p(X,Y) \wedge w(X) \wedge \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge w(X) \wedge m(T)] \quad \} \end{array}$$

We start the simplification process by applying $\mathsf{After}_{\mathcal{L}_H}$ *to S wrt U.*

$$\begin{array}{rll} \mathsf{After}^U_{\mathcal{L}_H}(S) \equiv \{ & \leftarrow & (m(X) \vee X = \mathbf{a}) \wedge w(X), \\ & \leftarrow & p(X,Y) \wedge (m(X) \vee X = \mathbf{a}) \wedge \\ & & \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge (m(X) \vee X = \mathbf{a}) \wedge w(T)], \\ & \leftarrow & p(X,Y) \wedge w(X) \wedge \\ & & \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge w(X) \wedge (m(T) \vee T = \mathbf{a})] \ \} \end{array}$$

After eliminating the disjunctions at level 0, $\mathsf{After}^U_{\mathcal{L}_H}(S)$ *is as follows:*

$$\begin{array}{l} \{\leftarrow m(X) \wedge w(X), \\ \ \leftarrow X = \mathbf{a} \wedge w(X), \\ \ \leftarrow p(X,Y) \wedge m(X) \wedge \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge (m(X) \vee X = \mathbf{a}) \wedge w(T)], \\ \ \leftarrow p(X,Y) \wedge X = \mathbf{a} \wedge \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge (m(X) \vee X = \mathbf{a}) \wedge w(T)], \\ \ \leftarrow p(X,Y) \wedge w(X) \wedge \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge w(X) \wedge (m(T) \vee T = \mathbf{a})] \ \} \end{array}$$

Now, we can eliminate the disjunctions at level 1 *and obtain the following set.*

$$\begin{array}{rl} \{\leftarrow & m(X) \wedge w(X), \\ \ \leftarrow & X = \mathbf{a} \wedge w(X), \\ \ \leftarrow & p(X,Y) \wedge m(X) \wedge \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge m(X) \wedge w(T)] \wedge \\ & \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge X = \mathbf{a} \wedge w(T)], \\ \ \leftarrow & p(X,Y) \wedge X = \mathbf{a} \wedge \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge m(X) \wedge w(T)] \wedge \\ & \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge X = \mathbf{a} \wedge w(T)], \\ \ \leftarrow & p(X,Y) \wedge w(X) \wedge \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge w(X) \wedge m(T)] \wedge \\ & \neg\exists(T,Z)[p(X,Z) \wedge p(T,Z) \wedge w(X) \wedge T = \mathbf{a}] \quad \} \end{array}$$

We can now proceed with the optimization of this set of extended denials by using the $\mathsf{Optimize}_{\mathcal{L}_H}$ *transformation. Clearly, the first, the third, and the fifth extended denial are extended-subsumed by the first, the second, and the third extended denial, respectively, in* $\mathsf{Unfold}_{\mathcal{L}_H}(S)$ *and are thus*

eliminated. The second denial reduces to $\leftarrow w(\mathbf{a})$. *In the fourth denial, the equality* $X = \mathbf{a}$ *at level* 0 *is eliminated, thus substituting* X *with* $\mathbf{a}$ *in the whole extended denial. We obtain the following.*

$$\begin{array}{ll} \{\leftarrow & w(\mathbf{a}), \\ \leftarrow & p(\mathbf{a}, Y) \wedge \neg\exists(T, Z)[p(\mathbf{a}, Z) \wedge p(T, Z) \wedge m(\mathbf{a}) \wedge w(T)] \wedge \\ & \neg\exists(T, Z)[p(\mathbf{a}, Z) \wedge p(T, Z) \wedge \mathbf{a} = \mathbf{a} \wedge w(T)] \qquad \} \end{array}$$

For the last extended denial, first we can eliminate the trivially succeeding equality $\mathbf{a} = \mathbf{a}$ *from the body of the second NEE. Then, we can consider that*

$$\leftarrow \neg\exists(T, Z)[p(\mathbf{a}, Z) \wedge p(T, Z) \wedge m(\mathbf{a}) \wedge w(T)]$$

extended-subsumes

$$\leftarrow p(\mathbf{a}, Y) \wedge \neg\exists(T, Z)[p(\mathbf{a}, Z) \wedge p(T, Z) \wedge w(T)]$$

so we can eliminate by subsumption factoring the subsuming part and leave the subsumed one. The simplification procedure for $\mathcal{L}_{\mathsf{H}}$ *applied to* S *and* U *returns the following result.*

$$\begin{array}{lll} \mathsf{Simp}^{U}_{\mathcal{L}_{\mathsf{H}}}(S) = \{ & \leftarrow w(\mathbf{a}), \\ & \leftarrow p(\mathbf{a}, Y) \wedge \neg\exists(T, Z)[p(\mathbf{a}, Z) \wedge p(T, Z) \wedge w(T)] \quad \} \end{array}$$

This coincides with the result given in [11], rewritten with our notation, with the only difference being that they do not assume disjointness of IDB and EDB, so, in the latter extended denial, they have the extra conjunct $\neg\exists V[married_to(\mathbf{a}, V)]$.

We conclude this section by discussing the potential for practical integration of the proposed technique into concrete systems. As is well known, integrity constraints represent properties of the data that should always be kept satisfied. As such, they are typically defined at database design time. However, when the nature of the constraints is more complex than the mere primary or foreign key constraints natively supported by relational databases, their maintenance is almost always delegated to the applications using the DBMS or, more rarely, to manually defined triggers residing in the database along the lines indicated in [13].

We observe that the reactive behavior of triggers is suitable for implementing integrity constraint maintenance policies, since they are able to respond with corrective reactions to potentially offending actions (such as the execution of updates that violate the constraints). Additionally, triggers are expressive enough so as to be able to include in their behavior any SQL query, possibly embedded in a procedural language, so that complex properties of the data are easily captured. Besides being expressive from the point of view of the language, triggers also offer suitable execution modes that would allow one to implement integrity maintenance both according to the usual "corrective" policy (first, the illegal update is executed, then the trigger reacts and checks the constraints, and finally, the update is undone) and to the "preventive" policy that we advocate here (first, check if the update is potentially illegal and, if so, do nothing). The corrective policy corresponds to the 'AFTER' execution mode of triggers, while the prevention policy can be obtained through 'BEFORE' triggers, available in all the main DBMSs.

The main drawback regarding trigger-based integrity constraint maintenance is that the DBMSs have no built-in way to automatically derive triggers that would implement a simplified integrity check for any upcoming update, which means that the database or application designers need to manually define proper triggers. Our Simp operator, instead, could be used on top of triggers, at database schema design time, to automatically produce,

for each relation r in the database, the simplified checks to be executed in the face of updates to r. Such simplified checks can then be translated into SQL queries installed within triggers responding to updates to r. The translation from the datalog form to SQL is a straightforward process that poses no particular technical challenge.

The use of triggers is but one of the ways in which the Simp operator could greatly extend the capabilities of a DBMS to automatize the checking process so as to fully support in DBMSs much more expressive kinds of integrity constraints than primary and foreign key constraints. Yet, there is more than augmenting the expressiveness of the constraints: our simplification procedure also allows for expressing complex parametric update patterns that go beyond the single addition/deletion/update of a record from the database. In particular, we have shown how to deal with complex transactional patterns that may be executed in an all-or-nothing fashion and whose effect on the satisfaction of the integrity constraints might even be complex to compute by hand, while Simp would make this part completely transparent. In particular, we envision a catalog of pre-designed update patterns (much in the same way as standard prepared statements in DBMSs) that users and applications can seamlessly use without having to pay any attention to integrity constraint maintenance, which could be completely taken care of by the system.

We also observe that the interest in integrity constraint maintenance and detection of constraint violations is very high even outside of classical relational database. In particular, the advent of different data models has, if anything, fortified the attention to these topics. A notable example is that of the so-called graph databases, whose representation of nodes and arcs typically comes with additional properties and labels that can be queried through very recently standardized languages such as GQL [93]. While the model of graph databases is different from the relational one, their structure is also very conveniently represented through datalog-like formalisms. This representation has allowed several new research efforts to target the incremental maintenance of complex data properties of graph data [94]. On top of this, triggers are also gaining momentum within the graph database community and are now part of the overall movement towards a standardization process [14], which, together with the availability of standard techniques for translating datalog notation to graph query languages, provides an even stronger basis for the widespread diffusion of well-founded techniques for integrity checking and maintenance even outside relational DBMSs.

As regards the experimental evaluation of our framework, we refrain from trying to quantitatively assess the effectiveness of the proposed simplification procedure against synthetically generated datasets, as such an analysis would lead to potentially very unreliable results, due to several reasons:

- The proposed operators work at the schema (program) level, not at the data level, so the synthetic generation would need to concern schemas (and update patterns) instead of datasets.
- Although a few efforts exist to generate schemas, such as [95], there is no proper benchmark that comes with the generation of integrity constraints or update patterns and, even more importantly, with a ground truth indicating the ideal kind of simplification that one should aim to obtain.
- Although data-independent measures of the quality of a simplification could be defined, there is no univocally correct way of doing this, as was shown in [15]. The "easy way" is to just count the remaining literals in the simplified formula, with no guarantee, however, that the obtained formula is indeed easier to check. To this end, the (non-)equality elimination steps shown in Definition 18 naturally follow in this direction by consistently reducing the number of literals in higher-level NEEs.

- Database schema design activities require expertise and finesse that would be annihilated by a synthetic generation, leading to results of questionable utility.
- Datasets would of course come into play when checking the (simplified) integrity constraints against data proper, and a comparison with other techniques at that level could certainly provide favorable time measurements each time an illegal update is encountered, since its execution and retraction would be completely avoided with our preventive approach. However, even ignoring all the previous considerations on the lack of suitable synthetically generated tests, the number and incidence of illegal updates in such tests could be increased at will, thereby unfairly magnifying the advantages of our technique as well.
- In light of all the above considerations, we have preferred to focus on the few examples existing in the literature of simplification procedures that work on constraints outside the (simpler) $\mathcal{L}_s$ language and show that, even in those ad hoc built cases, our general procedure produces similar or identical simplified checks (with the additional advantage of guaranteeing the possibility of a preventive approach).

6. Conclusions

We applied program transformation operators to the generation of simplified integrity constraints, targeting an expressive class termed extended denials, which includes negated existential quantifiers. We believe that this is an important class, as it encompasses very common dependencies such as tuple-generating dependencies and equality-generating dependencies.

An immediate application of our operators is an automated process that, upon requests from an application, communicates with the database and transparently carries out the required simplified integrity checking operations. The would imply benefits in terms of efficiency and could leverage a compiled approach, since simplifications can be generated at design time for the most common update patterns.

Although we used a logical notation, standard ways of translating integrity constraints into SQL exist. Further investigation is needed in order to handle additional language concepts of SQL like null values. In [13], Decker showed how to implement integrity constraint checking by translating first-order logic specifications into SQL triggers. The result of our transformations can be combined with similar translation techniques and thus integrated in an active database system, according to the idea of embedding integrity control in triggers. In this way, the advantages of declarativity are combined with the efficiency of execution.

Other possible enhancements of the proposed framework may be developed using statistical measures on the data, such as estimated relation sizes and cost measurements for performing join and union operations. Work in this area is closely related to methods for dynamic query processing, e.g., [96,97].

The proposed procedure is guaranteed to terminate, as we approximated entailment with rewrite rules based on resolution, subsumption, and replacement of specific patterns. While [15] discusses a few cases in which, besides termination, the procedure guarantees also completeness, it would be interesting to pinpoint more specifically what is left out in more expressive cases. A careful evaluation of the effectiveness of a simplification procedure for integrity checking is still an open problem: besides the lack, even in theory, of a univocal definition of "perfect" simplification, there is no benchmark specifically designed for this task and, even though some attempts exist at synthetically generating complex database schemata (as in [95]), these are not matched with update patterns and integrity constraints and, therefore, do not offer any ground truth to compare with. Additionally, the very interest and practical relevance of a synthetic schema generation, which is the typical result of a careful design by database experts, would be debatable at the very least.

Another line of research regards the combination of extended denials with other expressive scenarios, also individually described in [15], such as the addition of aggregates and arithmetic built-ins. While all these extensions could be trivially handled by a rule set comprising all rewrite rules defined for each specific scenario (such rules are mutually exclusive), it should be interesting to study whether further improvements can be obtained by exploiting the interaction between these rules. This is a limitation of our study which could be addressed by future work.

Funding: This research received no external funding.

Data Availability Statement: No datasets were used for the purposes of this theoretical paper.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Nicolas, J.M. Logic for Improving Integrity Checking in Relational Data Bases. *Acta Inform.* **1982**, *18*, 227–253. [CrossRef]
2. Bernstein, P.A.; Blaustein, B.T. Fast Methods for Testing Quantified Relational Calculus Assertions. In Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data, Orlando, FL, USA, 2–4 June 1982; Schkolnick, M., Ed.; ACM Press: New York, NY, USA, 1982; pp. 39–50.
3. Henschen, L.; McCune, W.; Naqvi, S. Compiling Constraint-Checking Programs from First-Order Formulas. In Proceedings of the Advances in Database Theory, Los Angeles, CA, USA, 1–5 February 1988; Gallaire, H., Minker, J., Nicolas, J.M., Eds.; Plenum Press: New York, NY, USA, 1984; Volume 2, pp. 145–169.
4. Hsu, A.; Imielinski, T. Integrity Checking for Multiple Updates. In Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, Austin, TX, USA, 28–31 May 1985; Navathe, S.B., Ed.; ACM Press: New York, NY, USA, 1985; pp. 152–168.
5. Lloyd, J.W.; Sonenberg, L.; Topor, R.W. Integrity Constraint Checking in Stratified Databases. *J. Log. Program.* **1987**, *4*, 331–343. [CrossRef]
6. Qian, X. An Effective Method for Integrity Constraint Simplification. In Proceedings of the Fourth International Conference on Data Engineering, Los Angeles, CA, USA, 1–5 February 1988; IEEE Computer Society: Washington, DC, USA, 1988; pp. 338–345.
7. Sadri, F.; Kowalski, R. A Theorem-Proving Approach to Database Integrity. In *Foundations of Deductive Databases and Logic Programming*; Minker, J., Ed.; Morgan Kaufmann: Los Altos, CA, USA, 1988; pp. 313–362.
8. Chakravarthy, U.S.; Grant, J.; Minker, J. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst. TODS* **1990**, *15*, 162–207. [CrossRef]
9. Decker, H.; Celma, M. A Slick Procedure for Integrity Checking in Deductive Databases. In *Logic Programming, Proceedings of the 11th International Conference on Logic Programming, Santa Margherita Ligure, Italy, 13–18 June 1994*; Van Hentenryck, P., Ed.; MIT Press: Cambridge, MA, USA, 1994; pp. 456–469.
10. Lee, S.Y.; Ling, T.W. Further Improvements on Integrity Constraint Checking for Stratifiable Deductive Databases. In Proceedings of the VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, Mumbai, India, 3–6 September 1996; Vijayaraman, T.M., Buchmann, A.P., Mohan, C., Sarda, N.L., Eds.; Morgan Kaufmann: San Francisco, CA, USA, 1996; pp. 495–505.
11. Leuschel, M.; de Schreye, D. Creating Specialised Integrity Checks Through Partial Evaluation of Meta-Interpreters. *J. Log. Program.* **1998**, *36*, 149–193. [CrossRef]
12. Seljée, R.; de Swart, H.C.M. Three Types of Redundancy in Integrity Checking: An Optimal Solution. *Data Knowl. Eng.* **1999**, *30*, 135–151. [CrossRef]
13. Decker, H. Translating advanced integrity checking technology to SQL. In *Database Integrity: Challenges and Solutions*; Doorn, J.H., Rivero, L.C., Eds.; Idea Group Publishing: Hershey, PA, USA, 2002; pp. 203–249.
14. Ceri, S.; Bernasconi, A.; Gagliardi, A.; Martinenghi, D.; Bellomarini, L.; Magnanimi, D. PG-Triggers: Triggers for Property Graphs. In Proceedings of the Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago, Chile, 9–15 June 2024; Barceló, P., Sánchez-Pi, N., Meliou, A., Sudarshan, S., Eds.; ACM: New York, NY, USA, 2024; pp. 373–385. [CrossRef]
15. Martinenghi, D. Advanced Techniques for Efficient Data Integrity Checking. Ph.D. Thesis, Department of Computer Science, Roskilde University, Roskilde, Denmark, 2005.
16. Christiansen, H.; Martinenghi, D. Simplification of Database Integrity Constraints Revisited: A Transformational Approach. In *Logic Based Program Synthesis and Transformation, Proceedings of the 13th International Symposium LOPSTR 2003, Uppsala, Sweden, 25–27 August 2003*; Lecture Notes in Computer Science; Revised Selected Papers; Bruynooghe, M., Ed.; Springer: Berlin/Heidelberg, Germany, 2003; Volume 3018, pp. 178–197. [CrossRef]

17. Chandra, A.K.; Merlin, P.M. Optimal implementation of conjunctive queries in relational databases. In Proceedings of the 9th Annual ACM Symposium on Theory of Computing, ACM, Boulder, CO, USA, 4–6 May 1977; pp. 77–90.
18. Shmueli, O. Decidability and expressiveness aspects of logic queries. In Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA, USA, 23–25 March 1987; pp. 237–249. [CrossRef]
19. Calì, A.; Martinenghi, D. Conjunctive Query Containment under Access Limitations. In Proceedings of the Conceptual Modeling—ER 2008, 27th International Conference on Conceptual Modeling, Barcelona, Spain, 20–24 October 2008; Lecture Notes in Computer Science; Proceedings; Li, Q., Spaccapietra, S., Yu, E.S.K., Olivé, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5231, pp. 326–340. [CrossRef]
20. Chomicki, J. Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Trans. Database Syst. TODS* **1995**, *20*, 149–186. [CrossRef]
21. Cowley, W.; Plexousakis, D. Temporal Integrity Constraints with Indeterminacy. In Proceedings of the VLDB 2000—26th International Conference on Very Large Data Bases, Cairo, Egypt, 10–14 September 2000; Abbadi, A.E., Brodie, M.L., Chakravarthy, S., Dayal, U., Kamel, N., Schlageter, G., Whang, K.Y., Eds.; Morgan Kaufmann: Los Altos, CA, USA, 2000; pp. 441–450.
22. Carmo, J.; Demolombe, R.; Jones, A.J.I. An Application of Deontic Logic to Information System Constraints. *Fundam. Inform.* **2001**, *48*, 165–181.
23. Godfrey, P.; Gryz, J.; Zuzarte, C. Exploiting constraint-like data characterizations in query optimization. In Proceedings of the SIGMOD '01: 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, CA, USA, 21–24 May 2001; Aref, W.G., Ed.; ACM Press: New York, NY, USA, 2001; pp. 582–592. [CrossRef]
24. Krr, N.; Zilberstein, S. Scoring-based methods for preference representation and reasoning. In Proceedings of the 21st National Conference on Artificial Intelligence (AAAI), Boston, MA, USA, 16–20 July 2006; pp. 539–545.
25. Brafman, R.I.; Domshlak, C. Preference handling: An AI perspective. *AI Mag.* **2006**, *28*, 58–68.
26. Parsons, S.; Wooldridge, M. Scoring functions for user preference modeling. In Proceedings of the ACM International Conference on Intelligent User Interfaces (IUI), San Francisco, CA, USA, 13–16 January 2002; pp. 19–26.
27. Faulkner, L.; Kersten, M.L. Preference handling in database systems. *VLDB J.* **2016**, *25*, 573–600.
28. Kiessling, W. Foundations of preferences in database systems. In Proceedings of the 28th International Conference on Very Large Databases (VLDB), Hong Kong, China, 20–23 August 2002; Morgan Kaufmann: Los Altos, CA, USA, 2002; pp. 311–322.
29. Chomicki, J. Preference formulas in relational queries. *ACM Trans. Database Syst. TODS* **2003**, *28*, 427–466. [CrossRef]
30. Agrawal, R.; Wimmers, E.L. Preference SQL: Flexible preference queries in databases. In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, Baltimore, MD, USA, 14–16 June 2005; pp. 560–572.
31. Bárány, V.; Benedikt, M.; Bourhis, P. Access patterns and integrity constraints revisited. In Proceedings of the Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, 18–22 March 2013; Tan, W., Guerrini, G., Catania, B., Gounaris, A., Eds.; ACM: New York, NY, USA, 2013; pp. 213–224. [CrossRef]
32. Codd, E.F. Further normalization of the database relational model. In Proceedings of the Courant Computer Science Symposium 6: Data Base Systems, New York, NY, USA, 24–25 May 1971; Rustin, R., Ed.; Prentice-Hall: Englewood Cliffs, NJ, USA, 1972; pp. 33–64.
33. Fagin, R. Multivalued Dependencies and a New Normal Form for Relational Databases. *ACM Trans. Database Syst. TODS* **1977**, *2*, 262–278. [CrossRef]
34. Fagin, R. Horn clauses and database dependencies. *J. ACM* **1982**, *29*, 952–985. [CrossRef]
35. Beeri, C.; Vardi, M.Y. A Proof Procedure for Data Dependencies. *J. ACM* **1984**, *31*, 718–741. [CrossRef]
36. Fagin, R.; Kolaitis, P.G.; Miller, R.J.; Popa, L. Data exchange: Semantics and query answering. *Theor. Comput. Sci.* **2005**, *336*, 89–124. [CrossRef]
37. Ullman, J.D. *Principles of Database and Knowledge-Base Systems, Volume I*; Computer Science Press: New York, NY, USA, 1988.
38. Ullman, J.D. *Principles of Database and Knowledge-Base Systems, Volume II*; Computer Science Press: New York, NY, USA, 1989.
39. Grefen, P.W.P.J. Combining Theory and Practice in Integrity Control: A Declarative Approach to the Specification of a Transaction Modification Subsystem. In Proceedings of the 19th International Conference on Very Large Data Bases, Dublin, Ireland, 24–27 August 1993; Agrawal, R., Baker, S., Bell, D.A., Eds.; Morgan Kaufmann: Los Altos, CA, USA, 1993; pp. 581–591.
40. Arenas, M.; Bertossi, L.E.; Chomicki, J. Consistent Query Answers in Inconsistent Databases. In Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, PA, USA, 31 May–2 June 1999; pp. 68–79.
41. Calì, A.; Lembo, D.; Rosati, R. On the decidability and complexity of query answering over inconsistent and incomplete databases. In Proceedings of the PODS '03: Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, New York, NY, USA, 9–12 June 2003; pp. 260–271. [CrossRef]

42. Arieli, O.; Denecker, M.; Nuffelen, B.V.; Bruynooghe, M. Database Repair by Signed Formulae. In Proceedings of the Foundations of Information and Knowledge Systems, Third International Symposium (FoIKS 2004), Wilhelminenburg Castle, Austria, 17–20 February 2004; Lecture Notes in Computer Science; Seipel, D., Torres, J.M.T., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; Volume 2942, pp. 14–30.
43. Bertossi, L.E. *Database Repairing and Consistent Query Answering*; Synthesis Lectures on Data Management; Morgan & Claypool Publishers: San Rafael, CA, USA, 2011. [CrossRef]
44. Bertossi, L.E. Database Repairs and Consistent Query Answering: Origins and Further Developments. In Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS, Amsterdam, The Netherlands, 30 June–5 July 2019; Suciu, D., Skritek, S., Koch, C., Eds.; ACM: New York, NY, USA, 2019; pp. 48–58. [CrossRef]
45. Decker, H.; Martinenghi, D. A Relaxed Approach to Integrity and Inconsistency in Databases. In Proceedings of the Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference, LPAR 2006, Phnom Penh, Cambodia, 13–17 November 2006; Lecture Notes in Computer Science; Proceedings; Hermann, M., Voronkov, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4246, pp. 287–301. [CrossRef]
46. Decker, H.; Martinenghi, D. Avenues to Flexible Data Integrity Checking. In Proceedings of the 17th International Workshop on Database and Expert Systems Applications (DEXA 2006), Krakow, Poland, 4–8 September 2006; IEEE Computer Society: Washington, DC, USA, 2006; pp. 425–429. [CrossRef]
47. Decker, H.; Martinenghi, D. Getting Rid of Straitjackets for Flexible Integrity Checking. In Proceedings of the 18th International Workshop on Database and Expert Systems Applications (DEXA 2007), Regensburg, Germany, 3–7 September 2007; IEEE Computer Society: Washington, DC, USA, 2007; pp. 360–364. [CrossRef]
48. Grant, J.; Hunter, A. Measuring inconsistency in knowledgebases. *J. Intell. Inf. Syst.* **2006**, *27*, 159–184. [CrossRef]
49. Grant, J.; Hunter, A. Measuring the Good and the Bad in Inconsistent Information. In Proceedings of the IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, 16–22 July 2011; Walsh, T., Ed.; IJCAI/AAAI: Montreal, QC, Canada, 2011; pp. 2632–2637. [CrossRef]
50. Grant, J.; Hunter, A. Distance-Based Measures of Inconsistency. In Proceedings of the Symbolic and Quantitative Approaches to Reasoning with Uncertainty—12th European Conference, ECSQARU 2013, Utrecht, The Netherlands, 8–10 July 2013; Lecture Notes in Computer Science, Proceedings; van der Gaag, L.C., Ed.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7958, pp. 230–241. [CrossRef]
51. Grant, J.; Hunter, A. Analysing inconsistent information using distance-based measures. *Int. J. Approx. Reason.* **2017**, *89*, 3–26. [CrossRef]
52. Grant, J.; Hunter, A. Semantic inconsistency measures using 3-valued logics. *Int. J. Approx. Reason.* **2023**, *156*, 38–60. [CrossRef]
53. Christiansen, H.; Martinenghi, D. Simplification of Integrity Constraints for Data Integration. In Proceedings of the Foundations of Information and Knowledge Systems, Third International Symposium, FoIKS 2004, Wilhelminenberg Castle, Austria, 17–20 February 2004; Lecture Notes in Computer Science; Proceedings; Seipel, D., Torres, J.M.T., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; Volume 2942, pp. 31–48. [CrossRef]
54. Martinenghi, D. Simplification of Integrity Constraints with Aggregates and Arithmetic Built-Ins. In Proceedings of the Flexible Query Answering Systems, 6th International Conference, FQAS 2004, Lyon, France, 24–26 June 2004; Lecture Notes in Computer Science; Proceedings; Christiansen, H., Hacid, M., Andreasen, T., Larsen, H.L., Eds. Springer: Berlin/Heidelberg, Germany, 2004; Volume 3055, pp. 348–361. [CrossRef]
55. Samarin, S.D.; Amini, M. Integrity Checking for Aggregate Queries. *IEEE Access* **2021**, *9*, 74068–74084. [CrossRef]
56. Martinenghi, D.; Christiansen, H. Transaction Management with Integrity Checking. In Proceedings of the Database and Expert Systems Applications, 16th International Conference, DEXA 2005, Copenhagen, Denmark, 22–26 August 2005; Lecture Notes in Computer Science; Proceedings; Andersen, K.V., Debenham, J.K., Wagner, R.R., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3588, pp. 606–615. [CrossRef]
57. Christiansen, H.; Martinenghi, D. Symbolic Constraints for Meta-Logic Programming. *Appl. Artif. Intell.* **2000**, *14*, 345–367. [CrossRef]
58. Kenig, B.; Suciu, D. Integrity Constraints Revisited: From Exact to Approximate Implication. In Proceedings of the 23rd International Conference on Database Theory, ICDT 2020, Copenhagen, Denmark, 30 March–2 April 2020; LIPIcs; Lutz, C., Jung, J.C., Eds.; Schloss Dagstuhl-Leibniz-Zentrum für Informatik: Wadern, Germany, 2020; Volume 155, pp. 18:1–18:20. [CrossRef]
59. Yu, H.; Hu, Q.; Yang, Z.; Liu, H. Efficient Continuous Big Data Integrity Checking for Decentralized Storage. *IEEE Trans. Netw. Sci. Eng.* **2021**, *8*, 1658–1673. [CrossRef]
60. Yang, C.; Tao, X.; Wang, S.; Zhao, F. Data Integrity Checking Supporting Reliable Data Migration in Cloud Storage. In Proceedings of the Wireless Algorithms, Systems, and Applications—15th International Conference, WASA 2020, Qingdao, China, 13–15 September 2020; Lecture Notes in Computer Science; Proceedings, Part I; Yu, D., Dressler, F., Yu, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2020; Volume 12384, pp. 615–626. [CrossRef]

61. Masciari, E. Trajectory Clustering via Effective Partitioning. In Proceedings of the Flexible Query Answering Systems, 8th International Conference, FQAS 2009, Roskilde, Denmark, 26–28 October 2009; Lecture Notes in Computer Science; Proceedings; Andreasen, T., Yager, R.R., Bulskov, H., Christiansen, H., Larsen, H.L., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5822, pp. 358–370. [CrossRef]
62. Masciari, E.; Mazzeo, G.M.; Zaniolo, C. Analysing microarray expression data through effective clustering. *Inf. Sci.* **2014**, *262*, 32–45. [CrossRef]
63. Masciari, E.; Moscato, V.; Picariello, A.; Sperlì, G. A Deep Learning Approach to Fake News Detection. In Proceedings of the Foundations of Intelligent Systems—25th International Symposium, ISMIS 2020, Graz, Austria, 23–25 September 2020; Lecture Notes in Computer Science; Helic, D., Leitner, G., Stettinger, M., Felfernig, A., Ras, Z.W., Eds.; Springer: Berlin/Heidelberg, Germany, 2020; Volume 12117, pp. 113–122. [CrossRef]
64. Masciari, E.; Moscato, V.; Picariello, A.; Sperlì, G. Detecting fake news by image analysis. In Proceedings of the IDEAS 2020: 24th International Database Engineering & Applications Symposium, Seoul, Republic of Korea, 12–14 August 2020; Desai, B.C., Cho, W., Eds.; ACM: New York, NY, USA, 2020; pp. 27:1–27:5. [CrossRef]
65. Fazzinga, B.; Flesca, S.; Masciari, E.; Furfaro, F. Efficient and effective RFID data warehousing. In Proceedings of the International Database Engineering and Applications Symposium (IDEAS 2009), Cetraro, Italy, 16–18 September 2009; International Conference Proceeding Series; Desai, B.C., Saccà, D., Greco, S., Eds.; ACM: New York, NY, USA, 2009; pp. 251–258. [CrossRef]
66. Fazzinga, B.; Flesca, S.; Furfaro, F.; Masciari, E. RFID-data compression for supporting aggregate queries. *ACM Trans. Database Syst.* **2013**, *38*, 11. [CrossRef]
67. Masciari, E.; Gao, S.; Zaniolo, C. Sequential pattern mining from trajectory data. In Proceedings of the 17th International Database Engineering & Applications Symposium, IDEAS '13, Barcelona, Spain, 9–11 October 2013; Desai, B.C., Larriba-Pey, J.L., Bernardino, J., Eds.; ACM: New York, NY, USA, 2013; pp. 162–167. [CrossRef]
68. Galli, L.; Fraternali, P.; Martinenghi, D.; Tagliasacchi, M.; Novak, J. A Draw-and-Guess Game to Segment Images. In Proceedings of the 2012 International Conference on Privacy, Security, Risk and Trust, PASSAT 2012, and 2012 International Confernece on Social Computing, SocialCom 2012, Amsterdam, The Netherlands, 3–5 September 2012; IEEE Computer Society: Washington, DC, USA, 2012; pp. 914–917. [CrossRef]
69. Bozzon, A.; Catallo, I.; Ciceri, E.; Fraternali, P.; Martinenghi, D.; Tagliasacchi, M. A Framework for Crowdsourced Multimedia Processing and Querying. In Proceedings of the First International Workshop on Crowdsourcing Web Search, Lyon, France, 17 April 2012; CEUR Workshop Proceedings; 2012; Volume 842, pp. 42–47.
70. Loni, B.; Menendez, M.; Georgescu, M.; Galli, L.; Massari, C.; Altingövde, I.S.; Martinenghi, D.; Melenhorst, M.; Vliegendhart, R.; Larson, M. Fashion-focused creative commons social dataset. In Proceedings of the Multimedia Systems Conference 2013, MMSys '13, Oslo, Norway, 27 February–1 March 2013; pp. 72–77.
71. Costa, G.; Manco, G.; Masciari, E. Dealing with trajectory streams by clustering and mathematical transforms. *J. Intell. Inf. Syst.* **2014**, *42*, 155–177. [CrossRef]
72. Wang, H.; He, D.; Yu, J.; Xiong, N.N.; Wu, B. RDIC: A blockchain-based remote data integrity checking scheme for IoT in 5G networks. *J. Parallel Distrib. Comput.* **2021**, *152*, 1–10. [CrossRef]
73. Srivastava, S.S.; Atre, M.; Sharma, S.; Gupta, R.; Shukla, S.K. Verity: Blockchains to Detect Insider Attacks in DBMS. *arXiv* **2019**, arXiv:1901.00228.
74. Ji, Y.; Shao, B.; Chang, J.; Bian, G. Flexible identity-based remote data integrity checking for cloud storage with privacy preserving property. *Clust. Comput.* **2022**, *25*, 337–349. [CrossRef]
75. Bienvenu, M.; Bourgaux, C. Inconsistency Handling in Prioritized Databases with Universal Constraints: Complexity Analysis and Links with Active Integrity Constraints. *arXiv* **2023**, arXiv:2306.03523.
76. Nilsson, U.; Małuzyński, J. *Logic, Programming and Prolog*, 2nd ed.; John Wiley & Sons Ltd.: Hoboken, NJ, USA, 1995.
77. Lloyd, J. *Foundations of Logic Programming*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 1987.
78. Apt, K.R.; Blair, H.A.; Walker, A. Towards a Theory of Declarative Knowledge. In *Foundations of Deductive Databases and Logic Programming*; Minker, J., Ed.; Morgan Kaufmann: Los Altos, CA, USA, 1988; pp. 89–148.
79. Apt, K.R.; Bol, R.N. Logic Programming and Negation: A Survey. *J. Log. Program.* **1994**, *19/20*, 9–71. [CrossRef]
80. Przymusinski, T.C. On the declarative semantics of deductive databases and logic programming. In *Foundations of Deductive Databases and Logic Programming*; Minker, J., Ed.; Morgan Kaufmann: Los Altos, CA, USA, 1988; pp. 193–216.
81. Gelfond, M.; Lifschitz, V. Minimal Model Semantics for Logic Programming. In *Logic Programming: Proceedings of the Fifth Logic Programming Symposium*; Kowalski, R., Bowen, K., Eds.; MIT Press: Cambridge, MA, USA, 1988; pp. 1070–1080.
82. van Gelder, A.; Ross, K.; Schlipf, J.S. Unfounded sets and well-founded semantics for general logic programs. In Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 88), Austin, TX, USA, 21–23 March 1988; pp. 221–230.
83. Dijkstra, E.W. *A Discipline of Programming*; Prentice-Hall: Hoboken, NJ, USA, 1976.
84. Hoare, C. An axiomatic basis for computer programming. *Commun. ACM* **1969**, *12*, 576–580. [CrossRef]

85. Bol, R.N. Loop checking in partial deduction. *J. Log. Program.* **1993**, *16*, 25–46. [CrossRef]
86. Grant, J.; Minker, J. The impact of logic programming on databases. *Commun. ACM CACM* **1992**, *35*, 66–81. [CrossRef]
87. Grant, J.; Minker, J. Integrity Constraints in Knowledge Based Systems. In *Knowledge Engineering Vol II, Applications*; Adeli, H., Ed.; McGraw-Hill: New York, NY, USA, 1990; pp. 1–25.
88. Eisinger, N.; Ohlbach, H.J. Deduction Systems Based on Resolution. In *Handbook of Logic in Artificial Intelligence and Logic Programming—Vol 1: Logical Foundations*; Gabbay, D.M., Hogger, C.J., Robinson, J.A., Eds.; Clarendon Press: Oxford, UK, 1993; pp. 183–271.
89. Robinson, J.A. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM* **1965**, *12*, 23–41. [CrossRef]
90. Chang, C.L.; Lee, R.C. *Symbolic Logic and Mechanical Theorem Proving*; Academic Press: Cambridge, MA, USA, 1973.
91. Topor, R.W. Domain-Independent Formulas and Databases. *Theor. Comput. Sci.* **1987**, *52*, 281–306. [CrossRef]
92. Lloyd, J.W.; Topor, R.W. Making Prolog more Expressive. *J. Log. Program.* **1984**, *3*, 225–240. [CrossRef]
93. Information Technology—Database Languages—GQL. 2024. Available online: https://www.iso.org/standard/76120.html (accessed on 17 February 2025).
94. Magnanimi, D.; Bellomarini, L.; Ceri, S.; Martinenghi, D. Reactive Company Control in Company Knowledge Graphs. In Proceedings of the 39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, 3–7 April 2023; pp. 3336–3348. [CrossRef]
95. Baldazzi, T.; Sallinger, E. iWarded, 2022. Available online: https://github.com/joint-kg-labs/iWarded (accessed on 17 February 2025).
96. Cole, R.L.; Graefe, G. Optimization of Dynamic Query Evaluation Plans. In Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, MN, USA, 24–27 May 1994; Snodgrass, R.T., Winslett, M., Eds.; ACM Press: New York, NY, USA, 1994; pp. 150–160.
97. Seshadri, P.; Hellerstein, J.M.; Pirahesh, H.; Leung, T.Y.C.; Ramakrishnan, R.; Srivastava, D.; Stuckey, P.J.; Sudarshan, S. Cost-Based Optimization for Magic: Algebra and Implementation. In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, QC, Canada, 4–6 June 1996; Jagadish, H.V., Mumick, I.S., Eds.; ACM Press: New York, NY, USA, 1996; pp. 435–446.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.