IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 19 September 2022, accepted 6 November 2022, date of publication 17 November 2022,
date of current version 5 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3222786

RESEARCH ARTICLE

Cleaning Data With Selection Rules

TOON BOECKLING, GUY DE TRÉ, AND ANTOON BRONSELAER
Department of Telecommunications and Information Processing, Ghent University, 9000 Ghent, Belgium
Corresponding author: Toon Boeckling (toon.boeckling@ugent.be)

ABSTRACT In this paper, we propose and study a type of tuple-level constraint that arises from the selection operator σ of relational algebra and that closely resembles the concepts of tuple-level denial constraints. We call this type of constraint *selection rules* and study their concepts and properties in the setting of data consistency management. The main contribution of this paper is the study of rule implication with selection rules in order to solve the error localization problem by means of the set cover method. It turns out that rule implication can be applied more easily if the representation of selection rules is extended in order to allow gaps between attribute values. We show that the properties of selection rules allow to improve the performance of rule implication. Evaluation of our approach compared to HoloClean on four real-world datasets shows promising results. First, repair with selection rules is often faster and less memory-consumable than HoloClean, especially when the amount of work that rule implication has to do is limited. Second, in terms of precision and recall of error detection and correction, repair strategies with selection rules almost always outperform HoloClean.

INDEX TERMS Data quality, relational algebra, consistency, rules.

I. INTRODUCTION

Data quality has become a crucial part of data management over the past decades. Apart from tackling completeness, accuracy or currency problems, one of the main problems is to safeguard consistency of data [1], [2], [3], [4], [5]. A popular approach to handle this is the *rule-based* approach, where rules (or constraints) model how consistent data should look like. Examples of data quality rules are functional dependencies and relatives [6], [7], [8], [9], [10], inclusion dependencies [7], denial constraints [11] and edit rules [12], which all differ in terms of *expressiveness*. With this in mind, choosing the type(s) of rules that one can use poses a difficult, yet fundamental problem. On the one hand, types of rules that are more expressive are able to capture more types of errors. On the other hand, more expressive types of rules come with a higher complexity in terms of fundamental problems like discovery, implication and repair.

In this paper, we focus on *tuple-level constraints* in relational databases. Although more expressive types of rules exist, it has been stated that these types of rules can capture a large portion of inconsistencies in real-world datasets [13].

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Asif Naeem.

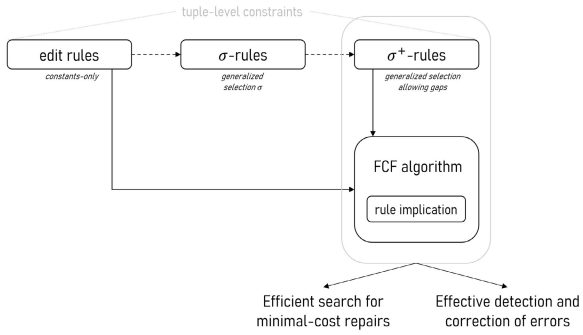

FIGURE 1. Schematic overview of the contributions presented in this paper.

More specifically, we investigate a type of tuple-level constraints that arises from the selection operator σ of relational algebra [14]. We call these constraints selection rules or, for short, σ-rules and show that these types of rules are a generalization of *edit rules* [12] and have the same expressive power as *tuple-level denial constraints* [11].

A schematic overview of our contributions presented in this paper is given in Figure 1. As a first contribution, we study the *concepts and properties* of σ-rules and investigate how the concepts and properties of edit rules and tuple-level denial

constraints relate to them. A second contribution is the proposal of an efficient algorithm for the *implication* of σ-rules based on these properties, in order to compose a so-called *sufficient set*. This algorithm will be an extension of the implication algorithms for regular edit rules [15], [16], [17] based on the implication procedure proposed by Fellegi & Holt [12]. The purpose of a sufficient set is to find minimal-cost repairs of inconsistent tuples easily. More precisely, if the cost of making a change to an attribute in a relation is fixed, then minimal-cost repairs can be found by searching for minimal set covers of failing rules in a sufficient set. Because of this property, repairing dirty data in a guaranteed minimal way is much faster than other procedures that rely on more expressive constraints, such as Llunatic [18] and HoloClean [19]. A remarkable result is that when domains are ordinal, the construction of a sufficient set is more difficult, because it is necessary to model gaps between attribute values to keep the number of implied rules manageable. The selection operator of relational algebra lacks the expressiveness to model such gaps easily. However, we propose a solution to this problem by extending the expressiveness of the selection σ to σ^+, where *gaps* between data values can be enforced that indicate how large the difference between two attribute values can(not) be. Interestingly, it does not matter whether we start from σ-rules or σ^+-rules: the implication procedure of both of them can be described in terms of σ^+-rules. As a third and final contribution, we assess the usefulness of selection rules by evaluating their capability to detect and correct erroneous data in real-world datasets.

The remainder of this paper is structured as follows. In Section II, we recall some basic concepts and notations related to the relational database model and relational algebra. In Section III, we formally define σ-rules and σ^+-rules, and state their concepts and properties. After this, in Section IV, we exploit these properties in a σ^+-rule implication algorithm that can be used to generate sufficient sets. In Section V, an overview is given of state-of-the-art literature related to the concepts and approaches studied in this paper, together with a positioning of our contributions within this literature. We evaluate our contributions in practical settings and discuss the results in Section VI. Finally, we state our conclusions and view on future research in Section VII.

II. PRELIMINARIES

In this paper, we assume the relational model for databases [14]. For a countable set of attributes $\mathcal{A}$, we denote for each $a \in \mathcal{A}$, the domain of a by A. For each $a \in \mathcal{A}$, we assume that the domain A is of one of the following types.

- Nominal: A is finite and features the equality relation $=$.
- Ordinal: A is equipped with an order $\leq$ and an order isomorphism exists from $(A, \leq)$ to $(\mathbb{Z}, \leq)$.
- Continuous: A is equipped with an order $\leq$ and operator $+$ and an order isomorphism exists from $(A, \leq, +)$ to $(\mathbb{R}, \leq, +)$.

When A is ordinal, it automatically comes with a successor function S. A successor function S defined on an ordinal domain A, applied on a value $v_i \in A$ (i.e., $S(v_i)$), returns an element $v_j \in A$, such that $v_i < v_j$ and $\nexists v_k \in A : v_i < v_k < v_j$. In words, a successor function S returns the closest element greater than its argument that exists in the domain on which S is defined. If we apply a successor function S, n times on a value $v \in A$, i.e.,

$$S^n(v) = \underbrace{S \circ S \circ \ldots \circ S(v)}_{n \text{ times}},$$

the n-th closest element greater than v in A is returned. If $n = 0$, the successor function S is *not* applied on value v and returns v, i.e., $S^0(v) = v$. Moreover, if $n < 0$, the inverse function is applied of which the outcome is the n-th closest element lower than v in A.

A *relation schema* $\mathcal{R} = \{a_1, \ldots, a_k\}$, or *schema* for short, is defined by a non-empty and finite subset of $\mathcal{A}$. A *relation* R over $\mathcal{R}$ is defined by a finite set $R \subseteq A_1 \times \ldots \times A_k$. Each element t of a relation R with schema $\mathcal{R}$ is called a *tuple* over $\mathcal{R}$. For any R with schema $\mathcal{R}$ and $X \subseteq \mathcal{R}$, we will denote the *projection* of R over X by $R[X]$. Note that, in this work, we always assume only one relation over one schema, but our contributions are not limited to a certain number of relations of schemata, because they can easily be applied when considering each of the relations/schemata separately.

A *propositional formula* φ over a schema $\mathcal{R} = \{a_1, \ldots, a_k\}$ is defined as a well-formed formula consisting of a set of *propositional predicates* and the Boolean operators $\wedge$, $\vee$ and $\neg$. Each propositional predicate (or predicate in short) P in φ is either

- a *constant* predicate $a\ \theta\ v$, where $a \in \mathcal{R}$, $v \in A$ and $\theta = \{\leq, <, =, \neq, >, \geq\}$ an operator on A, or
- a *variable* predicate $a_i\ \theta\ a_j$ where $a_i, a_j \in \mathcal{R}$ with $A_i = A_j$, and $\theta = \{\leq, <, =, \neq, >, \geq\}$ an operator on both A_i and A_j.

Evaluation of a predicate P for a given tuple t over $\mathcal{R}$ is then obtained by replacing the attributes in P with the projection of t over those attributes. In other words, for a tuple t, we test whether $t[a]\ \theta\ v$ (resp. $t[a_i]\ \theta\ t[a_j]$) is true or false. The *generalized selection* $\sigma_\varphi(R)$ with selection operator σ and propositional formula φ applied on relation R, is defined by $\{t \mid t \in R \wedge \varphi(t)\}$.

III. SELECTION RULES

A. BASIC DEFINITIONS AND PROPERTIES

Now the necessary concepts and notations related to the relational database model and relational algebra are recalled, we can formally introduce σ-rules.

Definition 1 (σ-Rule): A σ-rule over a schema $\mathcal{R}$ is defined (and denoted) by a propositional formula φ. A tuple t satisfies *(or* is consistent against*) the rule (denoted by $t \models \varphi$) if $\varphi(t)$ evaluates to false. Else, the tuple* fails *(or* is inconsistent against*) the rule (denoted by $t \not\models \varphi$).*

From Definition 1, it follows that each σ-rule describes a set of tuples that are not allowed in a consistent relation. If a σ-rule φ contains at least one variable predicate, it is called

TABLE 1. Operators and inverse operators.

θ	$\leq$	$<$	$=$	$\neq$	$>$	$\geq$
$\neg\theta$	$>$	$\geq$	$\neq$	$=$	$\leq$	$<$

TABLE 2. Example relation over stock schema.

tuple	PC	LTP	CID	CP
t_1	13.35	13.49	0.14	1.05
t_2	2.75	2.89	1.4	5.09
t_3	675.12	699.87	24.75	-3.67
t_4	-0.56	1.23	-26.83	-0.33
t_5	77.37	-65.13	-12.24	0

TABLE 3. Example σ-rules stock schema.

σ-rule	φ
φ^1	$\text{PC} < 0$
φ^2	$\text{LTP} < 0$
φ^3	$\text{PC} \leq \text{LTP} \wedge \text{CP} < 0$
φ^4	$\text{PC} \geq \text{LTP} \wedge \text{CP} > 0$
φ^5	$\text{PC} = \text{LTP} \wedge \text{CP} \neq 0$
φ^6	$\text{PC} \neq \text{LTP} \wedge \text{CP} = 0$
φ^7	$\text{CID} \geq 0 \wedge \text{CP} < 0$
φ^8	$\text{CID} \leq 0 \wedge \text{CP} > 0$
φ^9	$\text{CID} \neq 0 \wedge \text{CP} = 0$
φ^{10}	$\text{CID} = 0 \wedge \text{CP} \neq 0$

a *variable* σ-rule. Otherwise, it is called a *constant* σ-rule. As any propositional formula, φ can always be (re)written in disjunctive normal form (i.e. disjunction of conjunctions, DNF in short) and if the DNF of φ consists of more than one conjunctive clause (i.e., $\varphi = \varphi^1 \vee \ldots \vee \varphi^n$), we can rewrite φ as a set of σ-rules $\Phi = \{\varphi^1, \ldots, \varphi^n\}$. Doing so, a tuple t then fails Φ if any of the conjunctive clauses $\varphi^i \in \Phi$ evaluates to true. Furthermore, the set $\{\leq, <, =, \neq, >, \geq\}$ includes for each operator θ the inverse operator $\neg\theta$ so that Boolean negations of predicates can always be rewritten (cfr. Table 1). By these two facts, we can define a normal form of σ-rules, in which φ is restricted to a *conjunction* of predicates without negations. From now on, we assume that φ always has this normalized form, unless stated otherwise. Also, for easy notation, it is convenient to consider a third type of predicate, called a *set* predicate. A *set* predicate is of the form $a\ \theta\ V$, where $a \in \mathcal{R}$, $V \subseteq A$ and $\theta = \{\in, \notin\}$. It can easily be verified that when a σ-rule contains a set predicate, it can always be rewritten as a set of rules that do not use set predicates. Set predicates are therefore only a tool for convenience of notation.

Example 1: An example relation that we will use regularly in the remainder of this work is the relation over the stock schema[1] *given in Table 2. This relation consists of tuples representing the daily changes of stock prices. The schema consists of four attributes*[2] *with domain* $\mathbb{R}$*, which are previous close (PC, price of the stock on the latest daily transaction on the previous day), last trading price (LTP, price of the stock on the latest daily transaction on the current day), change in dollar (CID* $= LTP - PC$*) and change percentage (*$CP = \frac{CID*100}{PC}$*). Although it is not possible to represent all constraints on this relation exactly by means of* σ*-rules (e.g., the formula to calculate CP), a set of 10* σ*-rules (in normalized form) is given as an example in Table 3 to solve some consistency problems.*[3] *Examples are that it is not permitted that PC (resp. LTP) is strict negative (*φ^1*, resp.* φ^2*), that LTP is greater than or equal to PC and CP is strict negative (*φ^3*) or that CP equals zero and CID does not (*φ^9*).* φ^1*,* φ^2 *and* φ^7*-*φ^{10} *are constant* σ*-rules, whereas* φ^3*-*φ^6 *are variable* σ*-rules. In the data shown in Table 2,* t_1 *and* t_2 *are the only tuples that are consistent against the given set of rules, because* t_3 *fails* φ^3 *and* φ^7*,* t_4 *fails* φ^1 *and* φ^3*, and* t_5 *fails* φ^2*,* φ^6 *and* φ^9*. Note, however, that, although* t_2 *is consistent against the given set of rules, it does not follow the (linear) relationship to calculate CID and CP.*

In terms of expressiveness, σ-rules are a generalization of regular edit rules, introduced for categorical data in the framework of Fellegi and Holt [12] and extended to ordinal data in [17]. Indeed, each edit rule can be written as a set of (constant) σ-rules (and even one σ-rule if we allow set predicates), but the advantage of σ-rules, in terms of expressiveness, comes from the fact that one (variable) σ-rule can vouch for a set of infinitely many edit rules. Besides that, we can also state that σ-rules are a less expressive subclass of linear edit rules [20], because they are represented by linear (in)equalities involving one or two attributes with coefficients equal to 1. Also, σ-rules are a special case of denial constraints (DCs) where predicates are limited to a single tuple [4], [11], [21]. This makes σ-rules equivalent to tuple-level DCs and therefore, the concepts and properties of tuple-level DCs also apply for σ-rules. As we will explain in the following, the main virtue of these observations is that σ-rules come with a procedure for attribute elimination based on Fourier-Motzkin elimination, which helps to reduce the complexity of solving the error localization problem.

With this in mind, we will redefine the concepts and properties of edit rules and denial constraints within the setting of σ-rules. We say that an attribute is *involved* in a σ-rule if it appears in at least one of its predicates. The set of attributes involved in a σ-rule φ is denoted by $\mathcal{I}(\varphi)$. If the cardinality $|\mathcal{I}(\varphi)| = 1$, then the rule provides a constraint on the domain of that attribute and we call such rules *domain constraints*. Thereby, we will denote the set of predicates in φ involving (resp. not involving) an attribute a by means φ_a (resp. $\varphi_{\bar{a}}$). Also, the concepts of *redundancy*, *tautologies* and *contradictions* can be formally defined in the setting of σ-rules.

Definition 2 (Redundancy): A σ*-rule* φ^r *is redundant to a* σ*-rule* φ^d *(or* φ^d *dominates* φ^r*), both defined over* $\mathcal{R} = \{a_1, \ldots, a_k\}$*, if* $\varphi^r \Rightarrow \varphi^d$*. Semantically, this means that,* $\forall t \in A_1 \times \ldots \times A_k$*,* $t \not\models \varphi^r \Rightarrow t \not\models \varphi^d$*.*

[1] https://lunadong.com/fusionDataSets.htm

[2] Note that attributes change_in_dollar and change_percentage contain derived data, which is, strictly speaking, not allowed in a well-designed relational database. However, in this paper, we do allow it in order to make it more easy to explain certain concepts and techniques.

[3] For simplicity and clarity, this set does not necessarily contain all potential σ-rules that can be composed on the relation.

Definition 3 (Tautology): A σ-rule φ defined over $\mathcal{R} = \{a_1, \ldots, a_k\}$ is a (logical) tautology if $\varphi \equiv \bot$ (with $\bot$ representing 'always false'). Semantically, this means that, $\forall t \in A_1 \times \ldots \times A_k$, $t \models \varphi$.

Definition 4 (Contradiction): A σ-rule φ defined over $\mathcal{R} = \{a_1, \ldots, a_k\}$ is a (logical) contradiction if $\varphi \equiv \top$ (with $\top$ representing 'always true'). Semantically, this means that, $\forall t \in A_1 \times \ldots \times A_k$, $t \not\models \varphi$.

Note that a tautology is redundant to each σ-rule, and that each σ-rule is redundant to a contradiction.

Example 2: The values that attribute PC (resp. LTP) of the stock schema can take are limited by the domain constraint φ^1 (resp. φ^2). Indeed, $\mathcal{I}(\varphi^1) = \{PC\}$ (resp. $\mathcal{I}(\varphi^2) = \{LTP\}$). Moreover, $\mathcal{I}(\varphi^3) = \{PC, LTP, CP\}$. Besides that, $PC < -1$ is redundant to φ^1, $PC \leq 0 \wedge LTP \geq 0 \wedge CP < 0$ is redundant to φ^3 and $CID \geq 0 \wedge CP < 0 \wedge PC \leq LTP$ is redundant to φ^3 and φ^7. Finally, $PC \leq LTP \wedge PC > LTP$ and $CP < 0 \wedge CID > 0 \wedge CP > CID$ are logical tautologies.

B. MINIMAL-COST REPAIRS

Now σ-rules are formally defined and their related concepts and properties are stated, we can use them as a tool to detect and repair inconsistent tuples. In the remainder, the problem of finding correct repairs for inconsistent tuples is introduced, and we will propose a solution to this problem for σ-rules, which is in line with the solution introduced by Fellegi & Holt for regular edit rules [12].

To do this, suppose that a set of σ-rules Φ, defined over schema $\mathcal{R}$, and an inconsistent tuple t against Φ are given. By definition, t fails a set of σ-rules $\Phi' \subseteq \Phi$, i.e., $\forall \varphi \in \Phi'$, $t \not\models \varphi$. Any set of attributes $\mathcal{S} \subseteq \mathcal{R}$ for which different values can be assigned in t to construct t', such that t' is consistent against Φ, is called a *solution*. For this matter, we call t' a repair[4] of t based on solution $\mathcal{S}$. If weights are assigned to each attribute (e.g., to represent the cost of changing the value for this attribute), we can search for a solution with a minimal cost. A specific case is when all attributes receive equal weights, which implies solutions that are minimal in terms of set cardinality. Repairs based on minimal solutions are called minimal-cost repairs. Straightforwardly, a (minimal) solution $\mathcal{S}$ should contain at least one involved attribute for each failing rule in Φ'. Indeed, rules for which no involved attribute is added to $\mathcal{S}$ will remain failed after repair. We say that, if this condition is met, $\mathcal{S}$ is a (minimal) set cover of Φ' as it covers all failing rules and therefore, all (minimal) solutions are also (minimal) set covers.

Example 3: Assume that all attributes of the stock schema receive equal weights. For t_3 failing φ^3 and φ^7, the set $\{PC\}$ is not a correct solution, because we cannot construct a proper repair t' as φ^7 remains failed. The set $\mathcal{S} = \{PC, CP\}$ is a correct solution, because we can construct a repair t' by e.g., changing the value of PC to 675 and flipping the sign of CP. Moreover, the set $\mathcal{S}' = \{CP\} \subset \mathcal{S}$ is a minimal solution, because, to construct a proper repair t', flipping the sign of CP suffices and no solution with less attributes (i.e., an empty solution) exists (because t_3 is initially inconsistent).

Although the strategies to find minimal-cost repairs are quite simple and there is, initially, no guarantee that all (minimal) set covers are also (minimal) solutions (i.e., we cannot properly rely on the set cover method to find all correct (minimal) solutions), it comes with a great benefit in terms of finding solutions. Indeed, as we know for regular edit rules, it is guaranteed that all (minimal) set covers are also (minimal) solutions if we extend the initial set of rules Φ with *implied* rules [12]. Fortunately, because the properties of edit rules still hold in the setting of σ-rules, the same solution(s) can be applied.

Definition 5 (Implied σ-Rules): Given a set of σ-rules $\Phi_c = \{\varphi^1, \ldots, \varphi^n\}$ defined over $\mathcal{R} = \{a_1, \ldots, a_k\}$, a σ-rule φ^* over $\mathcal{R}$, for which

$$\varphi^* \Rightarrow \varphi^1 \vee \ldots \vee \varphi^n \tag{1}$$

is called an implied σ-rule, if φ^* is not a tautology. Semantically, this means that, $\forall t \in A_1 \times \ldots \times A_k$, $t \not\models \varphi^* \Rightarrow t \not\models \varphi^1 \vee \ldots \vee t \not\models \varphi^n$. We will call Φ_c a contributing set of φ^*.

Definition 5 provides a method based on propositional logic for validating whether a given σ-rule is implied or not. A procedure for generating implied rules from a given contributing set will be studied extensively in Section IV. Note, thereby, that each rule φ^r that is redundant to at least one $\varphi \in \Phi_c$ (with all $\varphi \in \Phi_c$ as special cases) is also an implied rule of Φ_c. With this made clear, it is straightforward to see from Definition 5 that implied rules do not forbid combinations of values that are not already forbidden by Φ_c, but they are indispensable in the sense that they guarantee correct error localization by means of the set cover method. Indeed, Fellegi & Holt proved, initially, that one needs to generate all implied rules in order to apply the set cover method properly [12], but generating all implied rules is a task of exponential complexity, because each combination of rules can potentially lead to the generation of many implied rules. Fortunately, the number of rules to imply can be limited, as implied rules are only *necessary* to adopt when they are (1) *new* and (2) *non-redundant* (NNR or necessary rules in short) [12], [15], [16], [22].

Definition 6 (New σ-Rules): An implied σ-rule φ^* defined over $\mathcal{R} = \{a_1, \ldots, a_k\}$, with contributing set $\Phi_c = \{\varphi^1, \ldots, \varphi^n\}$, is new if an attribute $a_g \in \mathcal{R}$ exists that is involved in each rule $\varphi \in \Phi_c$, but not in φ^*. We will call attribute a_g the generator of φ^*.

Definition 6 states that generating new rules comes down to a procedure of attribute (i.e., generator) elimination. The state-of-the-art algorithm for generating all necessary rules is the Field Code Forest (FCF) algorithm [15], [16]. This algorithm repeats two steps during execution, which are (1) the selection of a generator a_g and a corresponding set of rules Φ_{a_g} involving a_g (called candidate contributors), and (2) the generation of all necessary rules with generator a_g and contributing sets $\Phi_c \subseteq \Phi_{a_g}$. Eventually, the algorithm returns a set of rules that is *sufficient* to properly solve the

[4]We will not go into detail about repair strategies because this is out of scope.

error localization problem by means of the set cover method. Taking these notes into account, we will propose and study an algorithm to apply as the second step of FCF, specifically for σ-rules, in Section IV.

Example 4: The σ-rule $PC \neq LTP \wedge CID \geq 0 \wedge CP \leq 0$ defined over the stock schema, is an implied rule with contributing set $\{\varphi^6, \varphi^7\}$. The σ-rule $LTP \leq 0 \wedge CP > 0$ is a new rule with contributing set $\{\varphi^1, \varphi^4\}$. The generator of this latter rule is attribute PC.

C. EXTENSION TOWARDS σ^+-RULES

To end this section, we will extend the expressiveness of the selection operator σ, used by σ-rules (cfr. Definition 1), to σ^+. This extension should allow to enforce gaps between attribute values, modelling the smallest (e.g., $a_1 > a_2 + c$) or largest (e.g., $a_1 < a_2 + c$) prohibited difference c between the values of the two involved attributes a_1 and a_2. Regular edit rules, σ-rules and denial constraints lack the expressiveness to model such gaps easily. Take, for example, a constraint on the stock schema that only permits an increase of at least 1 from PC to LTP. Until now, at least one rule per value of $\mathbb{R}$ or infinitely many rules in total were needed to represent this (e.g., for value 0 of PC, a potential σ-rule that models this is $PC = 0 \wedge LTP < 1$). In the remainder, we will show that this constraint can be represented by means of only one σ^+-rule.

Definition 7 (σ^+-Rule): A σ^+-rule over a schema $\mathcal{R} = \{a_1, \ldots, a_k\}$ is a σ-rule in which each variable predicate in the propositional formula φ is of the form

- *$a_i \ \theta \ a_j$, in case $A_i = A_j$ are nominal domains, or*
- *$a_i \ \theta \ S^n(a_j)$, with $n \in \mathbb{Z}$, in case $A_i = A_j$ are ordinal domains with successor function S, or*
- *$a_i \ \theta \ a_j + c$, with c a constant, in case $A_i = A_j$ are continuous domains.*

For a tuple t over $\mathcal{R} = \{a_1, \ldots, a_k\}$, evaluation of a variable predicate of the form $a_i \ \theta \ S^n(a_j)$ (resp. $a_i \ \theta \ a_j + c$) with $a_i, a_j \in \mathcal{R}$, is done by testing whether $t[a_i] \ \theta \ S^n(t[a_j])$ (resp. $t[a_i] \ \theta \ t[a_j] + c$) is true or false. Variable predicates of the form $a_i \ \theta \ S^n(a_j)$ are called *successor predicates*.

With the definition of σ^+-rules set, we can make two interesting remarks. First, it is clear that the definition of σ^+-rules exactly matches the definition of σ-rules when domains are nominal. Second, each σ^+-rule can be written as a set of (sometimes infinitely many) σ-rules and each σ-rule can be converted to one σ^+-rule. Indeed, a σ-rule is a special kind of σ^+-rule in which each variable predicate is, depending on the domain type of A_i and A_j, of the form $a_i \ \theta \ a_j$, $a_i \ \theta \ S^0(a_j)$ or $a_i \ \theta \ a_j + 0$. A direct implication of this is that the concepts and properties of σ-rules can easily be transferred to the setting of σ^+-rules.

Example 5: All σ-rules given in Table 3 are also σ^+-rules (with $c = 0$ in the variable predicates). Now, we can use σ^+-rules to represent the example constraint which is stated in the beginning of this section and only permits an increase of at least 1 from attribute PC to attribute LTP by $PC > LTP - 1$.

IV. σ^+-RULE IMPLICATION

Now we have introduced two types of selection rules (σ and σ^+-rules), we can shift our attention to rule implication for these types of rules. Because σ-rules are a special subclass of σ^+-rules and, therefore, the properties of σ^+-rules also hold for σ-rules (cfr. III-C), we will mainly focus on implication of σ^+-rules.

A. THE IMPLICATION FUNCTION

Before going into detail about rule implication, we will first define an implication function that is fundamental in the process of attribute elimination. This function should allow to find all possible constraints implied by φ on all attributes (except 1) involved in φ.

Definition 8 (Implication Function): Given a propositional formula φ defined over $\mathcal{R} = \{a_1, \ldots, a_k\}$ and an attribute $a_g \in \mathcal{R}$, the implication function

$$Imp(\varphi, a_g) \tag{2}$$

returns a propositional formula, written as $\varphi^* = \varphi^{*1} \vee \ldots \vee \varphi^{*m}$ in DNF, that captures all φ^{*i} (with $1 \leq i \leq m$) such that $\varphi \Rightarrow \varphi^*$ and φ^* does not involve a_g.

With this implication function set, an important question that remains to be answered is how to construct φ^* as the result of $\text{Imp}(\varphi, a_g)$ correctly and completely. To answer this question, we can point out some observations. First, we can rely on the transitivity property of the operators $\{\leq, <, =, \neq, >, \geq\}$ for finding all possible constraints implied by φ. Indeed, exploiting this property for all *pairs* of predicates that are connected by a conjunction in φ, features a set of predicates representing these (potentially hidden) constraints. As an example, the conjunction of predicates $\{a_i \leq a_g, a_g \leq a_j\}$, with $A_i, A_j = \mathbb{R}$ reveals a constraint $a_i \leq a_j$ on both attributes a_i and a_j, in which a_g is eliminated, because $a_i \leq a_g \wedge a_g \leq a_j \Rightarrow a_i \leq a_j$. Second, we can state that exploiting the transitivity property on a pair of predicates may imply a propositional formula consisting of the conjunction of more than one (and potentially infinitely many) predicate(s). However, this formula automatically reduces, such that only the most restrictive implied predicate remains. Following the previous example, $a_i \leq a_g \wedge a_g \leq a_j \Rightarrow \bigwedge_{c \geq 0} a_i \leq a_j + c$, but because $\bigwedge_{c \geq 0} a_i \leq a_j + c \equiv a_i \leq a_j$, the only (and most restrictive) implied predicate that follows from this is $a_i \leq a_j$. Now, with this made clear, note that the maximal number of (hidden) predicates not involving a_g that are generated by the implication function, with $\varphi = \varphi^1 \vee \ldots \vee \varphi^n$ in DNF, for each $\varphi^i \in \varphi$ with k_i predicates, equals $\binom{k_i}{2}$, which grows exponentially with increasing k_i.

In Table 4 and 5, we have provided an overview of the most restrictive predicates, not involving generator a_g, that are constructed after exploiting the transitivity property on each possible pair of predicates connected by a conjunction (depending on the operators), both involving a_g. Note that Table 4 accounts for ordinal domains and Table 5 accounts for continuous domains and (in limited version) for nominal domains. Each cell in these tables contains two predicates,

depending on the fact whether we start from a pair of variable predicates (e.g., $a_g < a_i + c_1 \wedge a_g = a_j + c_2 \Rightarrow a_i > a_j + c_2 - c_1$) or from a pair consisting of one variable and one constant predicate (e.g., $a_g < a_i + c_1 \wedge a_g = v \Rightarrow a_i > v - c_1$). $\top$ represents a tautology, following from the fact that exploiting the transitivity property on the corresponding pairs of predicates will not reveal any more restrictive constraints on the involved attributes other than a_g. For example, $a_g < a_i + c_1 \wedge a_g < v$ only implies that a_i can take any value in its domain. An interesting observation is that exploiting the transitivity property on the pairs $\{a_g < S^m(a_i), a_g > S^n(a_j)\}$ and $\{a_g > S^m(a_i), a_g < S^n(a_j)\}$ results in predicates involving a_i and a_j with a 'self-expanding' gap (indicated by resp. $+1$ and -1 in the resulting predicates in Table 4). Indeed, if we consider these pairs, the constraint on both a_i and a_j should account for the fact that the gap between these attributes should be large enough such that a_g can take values meeting the original constraints in the pairs. For predicates involving attributes with continuous domains, this is not a problem, as there always exists a value for a_g meeting these original constraints. Note, hereby, that for some σ-rules involving attributes with ordinal domains, the result potentially contains an infinite number of σ-rules, because these gaps cannot be represented. This reveals an additional and important benefit of introducing σ^+-rules for rule implication. Finally, note that two trivial cases are not listed in the tables, which are the following. If a pair consists of any predicate that does not involve a_g, this predicate is automatically implied (e.g., $a_g < a_i \wedge a_j > 0 \Rightarrow a_j > 0$). If a pair consists of two predicates involving no other attribute than a_g, applying the transitivity property results in either a tautology (e.g., $a_g \geq 0 \wedge a_g \leq 1 \Rightarrow \top$) or a contradiction (e.g., $a_g \leq 0 \wedge a_g \geq 1 \Rightarrow \bot$).

B. GENERATING NECESSARY σ^+-RULES

As explained in III-B, generating necessary rules is crucial in order to properly execute the FCF algorithm for generating a sufficient set. Because we have generalized edit rules to σ^+-rules, we need an extension of the implication procedure proposed in [12] that works correctly and completely for (contributing) sets of σ^+-rules. In order to generate all *new* rules from a given set of σ^+-rules by means of generator a_g, we can rely on the following theorem.

Theorem 1: Consider a set of σ^+-rules $\Phi_{a_g} = \{\varphi^1, \ldots, \varphi^n\}$ that are defined over $\mathcal{R} = \{a_1, \ldots, a_k\}$ and involve attribute $a_g \in \mathcal{R}$. Applying

$$\neg Imp(\varphi, a_g), \tag{3}$$

with $\varphi = \neg\varphi^1 \wedge \ldots \wedge \neg\varphi^n$, results in a propositional formula, written as $\varphi^* = \varphi^{*_1} \vee \ldots \vee \varphi^{*_m}$ in DNF, capturing all new σ^+-rules φ^{*_i} (with $1 \leq i \leq m$) that can be generated with any $\Phi_c \subseteq \Phi_{a_g}$ as contributing set and a_g as generator.

Proof: See Appendix. □

Example 6: Consider again the σ-rules (or σ^+-rules) defined on the stock schema (cfr. Table 3). As we validated in III-B, the contributing set $\{\varphi^1, \varphi^4\}$ and generator PC can be used to construct a new σ-rule $LTP \leq 0 \wedge CP > 0$. Indeed,

$$\begin{aligned}
&\neg Imp(\neg\varphi^1 \wedge \neg\varphi^4, PC)\\
&\equiv \neg Imp(PC \geq 0 \wedge (PC < LTP \vee CP \leq 0), PC)\\
&\equiv \neg Imp((PC \geq 0 \wedge PC < LTP) \vee (PC \geq 0 \wedge CP \leq 0), PC)\\
&\equiv \neg(LTP > 0 \vee CP \leq 0)\\
&\equiv\ LTP \leq 0 \wedge CP > 0
\end{aligned}$$

Remark that passing any superset of $\{\varphi^1, \varphi^4\}$ as first argument to the implication function will also result in a propositional formula φ^* in which one of the conjunctive clauses φ^{*_i} equals $LTP \leq 0 \wedge CP > 0$.

Now, we point out three important remarks following from the construction procedure given in Theorem 1. First, if you want to generate all implied rules (including rules that are not new) from a set Φ, you can apply Theorem 1 and keep all resulting predicates (so not only the ones that do not involve a_g). We will denote this by $\text{Imp}(\varphi, -)$ in the following. Second, repeatedly applying Theorem 1, together with the observations listed above, ensures that all necessary σ^+-rules certainly will be generated, as stated in the beginning of this section. The reason for this is that the proposed implication procedure is a special case of Fourier-Motzkin elimination to eliminate attributes in *linear* edit rules [23], [24], [25]. Fellegi & Holt proved that repeatedly applying Fourier-Motzkin elimination on pairs of linear edit rules from a given set eventually results in a complete (and thus, sufficient) set [12]. In order to avoid a repeated application of this method, the proposed optimizations to generate sufficient sets (e.g., the FCF algorithm) can also be used [15], [16], [20], [22]. Third, note that it might be the case that the result of the proposed procedure contains (many) new rules that are redundant to other rules generated during executing the FCF algorithm. To resolve this, it is possible to test each generated rule for being redundant to all other rules on each completion of the implication procedure and treat redundant rules properly as described in [16].

To end this section, we will state the following corollary of Theorem 1, which gives the upper bound on the number of new σ^+-rules that can be generated when applying the proposed procedure.

Corollary 1: Consider a set of σ^+-rules $\Phi_{a_g} = \{\varphi^1, \ldots, \varphi^n\}$ that are defined over $\mathcal{R} = \{a_1, \ldots, a_k\}$ and involve attribute $a_g \in \mathcal{R}$. The maximal number of new σ^+-rules captured by the result of $\neg Imp(\varphi, a_g)$, with $\varphi = \neg\varphi^1 \wedge \ldots \wedge \neg\varphi^n$, is

$$\binom{n}{2}^{|\varphi^1|\cdot\ldots\cdot|\varphi^n|}, \tag{4}$$

with $|\varphi^i|$ the number of predicates in φ^i.

Proof: See Appendix. □

With this corollary stated, it is clear that it is sufficient to apply Theorem 1 in order to generate all necessary σ^+-rules with a given generator a_g (i.e., to adopt as the second step of the FCF algorithm). However, the number of generated new

TABLE 4. Overview of the most restrictive predicates, not involving a_g, constructed after exploiting the transitivity property on each possible pair of predicates, involving a_g. This overview accounts for ordinal domains.

$\wedge$	$a_g < S^n(a_j)$ $a_g < v$	$a_g \leq S^n(a_j)$ $a_g \leq v$	$a_g \neq S^n(a_j)$ $a_g \neq v$	$a_g = S^n(a_j)$ $a_g = v$	$a_g \geq S^n(a_j)$ $a_g \geq v$	$a_g > S^n(a_j)$ $a_g > v$
$a_g < S^m(a_i)$	$\top$ $\top$	$\top$ $\top$	$\top$ $\top$	$a_i > S^{n-m}(a_j)$ $a_i > S^{-m}(v)$	$a_i > S^{n-m}(a_j)$ $a_i > S^{-m}(v)$	$a_i > S^{n-m+1}(a_j)$ $a_i > S^{1-m}(v)$
$a_g \leq S^m(a_i)$	$\top$ $\top$	$\top$ $\top$	$\top$ $\top$	$a_i \geq S^{n-m}(a_j)$ $a_i \geq S^{-m}(v)$	$a_i \geq S^{n-m}(a_j)$ $a_i \geq S^{-m}(v)$	$a_i > S^{n-m}(a_j)$ $a_i > S^{-m}(v)$
$a_g \neq S^m(a_i)$	$\top$ $\top$	$\top$ $\top$	$\top$ $\top$	$a_i \neq S^{n-m}(a_j)$ $a_i \neq S^{-m}(v)$	$\top$ $\top$	$\top$ $\top$
$a_g = S^m(a_i)$	$a_i < S^{n-m}(a_j)$ $a_i < S^{-m}(v)$	$a_i \leq S^{n-m}(a_j)$ $a_i \leq S^{-m}(v)$	$a_i \neq S^{n-m}(a_j)$ $a_i \neq S^{-m}(v)$	$a_i = S^{n-m}(a_j)$ $a_i = S^{-m}(v)$	$a_i \geq S^{n-m}(a_j)$ $a_i \geq S^{-m}(v)$	$a_i > S^{n-m}(a_j)$ $a_i > S^{-m}(v)$
$a_g \geq S^m(a_i)$	$a_i < S^{n-m}(a_j)$ $a_i < S^{-m}(v)$	$a_i \leq S^{n-m}(a_j)$ $a_i \leq S^{-m}(v)$	$\top$ $\top$	$a_i \leq S^{n-m}(a_j)$ $a_i \leq S^{-m}(v)$	$\top$ $\top$	$\top$ $\top$
$a_g > S^m(a_i)$	$a_i < S^{n-m-1}(a_j)$ $a_i < S^{-m-1}(v)$	$a_i < S^{n-m}(a_j)$ $a_i < S^{-m}(v)$	$\top$ $\top$	$a_i < S^{n-m}(a_j)$ $a_i < S^{-m}(v)$	$\top$ $\top$	$\top$ $\top$

TABLE 5. Overview of the most restrictive predicates, not involving a_g, constructed after exploiting the transitivity property on each possible pair of predicates, involving a_g. This overview accounts for continuous domains. For nominal domains, only consider operators $=$ and $\neq$ and $c_1, c_2 = 0$.

$\wedge$	$a_g < a_j + c_2$ $a_g < v$	$a_g \leq a_j + c_2$ $a_g \leq v$	$a_g \neq a_j + c_2$ $a_g \neq v$	$a_g = a_j + c_2$ $a_g = v$	$a_g \geq a_j + c_2$ $a_g \geq v$	$a_g > a_j + c_2$ $a_g > v$
$a_g < a_i + c_1$	$\top$ $\top$	$\top$ $\top$	$\top$ $\top$	$a_i > a_j + c_2 - c_1$ $a_i > v - c_1$	$a_i > a_j + c_2 - c_1$ $a_i > v - c_1$	$a_i > a_j + c_2 - c_1$ $a_i > v - c_1$
$a_g \leq a_i + c_1$	$\top$ $\top$	$\top$ $\top$	$\top$ $\top$	$a_i \geq a_j + c_2 - c_1$ $a_i \geq v - c_1$	$a_i \geq a_j + c_2 - c_1$ $a_i \geq v - c_1$	$a_i > a_j + c_2 - c_1$ $a_i > v - c_1$
$a_g \neq a_i + c_1$	$\top$ $\top$	$\top$ $\top$	$\top$ $\top$	$a_i \neq a_j + c_2 - c_1$ $a_i \neq v - c_1$	$\top$ $\top$	$\top$ $\top$
$a_g = a_i + c_1$	$a_i < a_j + c_2 - c_1$ $a_i < v - c_1$	$a_i \leq a_j + c_2 - c_1$ $a_i \leq v - c_1$	$a_i \neq a_j + c_2 - c_1$ $a_i \neq v - c_1$	$a_i = a_j + c_2 - c_1$ $a_i = v - c_1$	$a_i \geq a_j + c_2 - c_1$ $a_i \geq v - c_1$	$a_i > a_j + c_2 - c_1$ $a_i > v - c_1$
$a_g \geq a_i + c_1$	$a_i < a_j + c_2 - c_1$ $a_i < v - c_1$	$a_i \leq a_j + c_2 - c_1$ $a_i \leq v - c_1$	$\top$ $\top$	$a_i \leq a_j + c_2 - c_1$ $a_i \leq v - c_1$	$\top$ $\top$	$\top$ $\top$
$a_g > a_i + c_1$	$a_i < a_j + c_2 - c_1$ $a_i < v - c_1$	$a_i < a_j + c_2 - c_1$ $a_i < v - c_1$	$\top$ $\top$	$a_i < a_j + c_2 - c_1$ $a_i < v - c_1$	$\top$ $\top$	$\top$ $\top$

rules can rapidly become unmanageable when the number of (predicates in the) candidate contributors increases. For two or three candidate contributors, this is not directly a problem, but when considering, for example, the five rules defined on the stock schema (cfr. Table 3) involving PC and generator PC as input parameters, the upper bound on the number of new rules with generator PC equals 10^8. The main reason for this is that the procedure will not exploit any strategy to avoid generating rules that are new, but not necessary (i.e., redundant). In the following section, we will investigate techniques in order to overcome this problem.

C. A σ^+-RULE IMPLICATION ALGORITHM

To overcome the potential performance problem of using the implication function when searching for all necessary rules, we will investigate some useful properties of σ^+-rule implication (cfr. IV-C1) and exploit these properties in an optimized σ^+-rule implication algorithm (cfr. IV-C2).

1) PROPERTIES

σ^+-rules come with some useful properties that can be exploited during the implication algorithm.

Proposition 1: Consider

- *two σ^+-rules φ^r and φ^d, defined over $\mathcal{R}$, and*
- *two contributing sets $\Phi_r = \{\varphi^r, \varphi\}$ and $\Phi_d = \{\varphi^d, \varphi\}$, with φ any σ^+-rule, defined over $\mathcal{R}$, which lead to the generation of resp. φ^{*_r} and φ^{*_d}, i.e.,*
  - ➢ $\varphi^{*_r} \equiv \neg Imp(\neg\varphi^r \wedge \neg\varphi, -)$, *and*
  - ➢ $\varphi^{*_d} \equiv \neg Imp(\neg\varphi^d \wedge \neg\varphi, -)$.

If $\varphi^r \Rightarrow \varphi^d$, then $\varphi^{*_r} \Rightarrow \varphi^{*_d}$.

Proof: See Appendix. □

Proposition 1 states that a contributing set Φ_r that contains a redundant rule φ^r will always lead to the generation of rules φ^{*_r} that are redundant. This implies that we should never test combinations of rules containing redundant rules during the implication algorithm, reducing the number of combinations to test. Moreover, due to this fact and the fact that redundant rules are not necessary, we can always ignore rules that are redundant to one of the candidate contributors and that are captured in the result of the implication function. Note that tautologies are special cases in this regard, because they are, by definition, redundant to all σ^+-rules.

Example 7: Consider again the σ-rules (or σ^+-rules) defined on the stock schema (cfr. Table 3), with $\varphi = \varphi^1$ and $\varphi^d = \varphi^4$. Moreover, consider an additional rule $\varphi^r = PC > LTP \wedge CP > 0$, such that $\varphi^r \Rightarrow \varphi^d$. Now, applying $\neg Imp(\neg\varphi^d \wedge \neg\varphi, -)$ results in $\varphi^{*_d} = \varphi \vee \varphi^d \vee (LTP \leq 0 \wedge CP > 0)$ and applying $\neg Imp(\neg\varphi^r \wedge \neg\varphi, -)$ results in $\varphi^{*_r} = \varphi \vee \varphi^r \vee (LTP < 0 \wedge CP > 0)$, such that $\varphi^{*_r} \Rightarrow \varphi^{*_d}$ (cfr. Proposition 1).

Proposition 2: Consider two σ^+-rules φ^1 and φ^2, defined over $\mathcal{R} = \{a_1, \ldots, a_k\}$. If $\varphi^1_{a_g} \Rightarrow \varphi^2_{a_g}$ (with $a_g \in \mathcal{R}$), then $\neg Imp(\neg\varphi^1 \wedge \neg\varphi^2, -)$ results in $\varphi^* = \varphi^{*_1} \vee \ldots \vee \varphi^{*_m}$ with for each φ^{*_i}

- *$\varphi^{*_i} \Rightarrow \varphi^1$, or*

- $\varphi^{*_i} \Rightarrow \varphi^2$, *or*
- φ^{*_i} *a new rule with a generator* $a_{g'} \in \mathcal{R}$ *other than* a_g.

Proof: See Appendix. □

Proposition 2 states that we should only test combinations of rules in which each σ^+-rule contributes to the elimination of a_g, again reducing the number of combinations to test. Concretely, this means that for any two rules φ^1 and φ^2 in a combination, $\varphi^1_{a_g} \not\Rightarrow \varphi^2_{a_g}$ and $\varphi^2_{a_g} \not\Rightarrow \varphi^1_{a_g}$. Otherwise, the combination will either lead to the construction of redundant rules or new rules with a generator $a_{g'}$ other than a_g, in which we are not interested. A first special case in this regard is when $\varphi^1 \Rightarrow \varphi^2$ (or $\varphi^2 \Rightarrow \varphi^1$), which automatically implies that $\varphi^1_{a_g} \Rightarrow \varphi^2_{a_g}$ (or $\varphi^2_{a_g} \Rightarrow \varphi^1_{a_g}$). This means that combinations should not contain pairs of rules in which one rule is redundant to the other. A second special case is when a_g is not involved in φ^1 (or φ^2), because other rules will never contribute to the elimination of a_g in combination with such rules. First, this means that only combinations of rules in which the generator is involved (i.e., candidate contributors) should be tested, which confirms the necessary condition in the definition of new rules (cfr. Definition 6). Second, once a combination of rules leads to the generation of a new rule φ^* with generator a_g, we should never add φ^* in a combination to test for leading to the generation of new rules with generator a_g.

Example 8: Consider the σ-rules φ^7 and φ^{10} defined on the stock schema and listed in Table 3. For this pair of rules, it is the case that $\varphi^7_{CP} \Rightarrow \varphi^{10}_{CP}$ and $\varphi^{10}_{CID} \Rightarrow \varphi^7_{CID}$, such that the result of applying $\neg Imp(\neg\varphi^7 \wedge \neg\varphi^{10}, -)$ will only contain σ-rules matching one of the options stated in Proposition 2. Indeed, $\neg Imp(\neg\varphi^7 \wedge \neg\varphi^{10}, -) \equiv \varphi^7 \vee \varphi^{10}$ (cfr. Proposition 2).

2) ALGORITHM

Below, we will propose a σ^+-rule implication algorithm exploiting the properties stated above in order to reduce the number of combinations to test, which serves as an alternative to the general implication procedure (cfr. IV-B) in case this procedure may result in a large number of implied σ^+-rules. As we already mentioned in the previous, the algorithm will be a generalization of the implication algorithm for regular edit rules, proposed in [17]. Specifically, the algorithm will keep a list of implied σ^+-rules that still can lead to the generation of a necessary rule by combining it, in an optimized way (i.e., by exploiting Proposition 1 and Proposition 2), with one or more other σ^+-rules. If it is certain that a rule cannot lead in any case to the construction of a necessary rule, the rule is ignored. The pseudocode of the σ^+-rule implication algorithm is given in Algorithm 1.

As first input parameter, the algorithm expects a generator a_g for which necessary rules will be generated. As second input parameter, the algorithm expects any set Φ_{a_g} consisting of candidate contributors for a_g. Combinations of rules in Φ_{a_g} will be tested for contributing to the generation of necessary rules with generator a_g. From Proposition 2, we know that no other rules (i.e., rules in which a_g is not involved) should be considered as they will not account for this matter. In the first step of the algorithm, it is verified whether the set of selected candidate contributors can eventually lead to the elimination of a_g (line 2). If this is not the case, an empty set is returned (line 3). Then, before searching for necessary rules, three variables are initialized (line 4-6), which are used throughout the execution and which are listed below.

Algorithm 1 σ^+-Rule Implication Algorithm

```
1:  function getNecessaryRules(a_g, Φ_{a_g})
2:      if ⋁_{φ∈Φ_{a_g}} φ_{a_g} ≢ ⊤ then
3:          return ∅
4:      ℚ ← [φ | φ ∈ Φ_{a_g}]
5:      Φ_c ← {φ : {φ} | φ ∈ Φ_{a_g}}
6:      Φ_* ← ∅
7:      while ℚ ≠ ∅ do
8:          φ^q ← ℚ.dequeue()
9:          Φ_q ← Φ_c[φ^q]
10:         for all φ^k ∈ Φ_{a_g} \ Φ_q do
11:             if φ^k_{a_g} ⇒ φ^q_{a_g} ∨ φ^q_{a_g} ⇒ φ^k_{a_g} then
12:                 continue
13:             Φ_{*_q} ← ¬Imp(¬φ^q ∧ ¬φ^k, −)
14:             for all φ^{*_q} ∈ Φ_{*_q} do
15:                 if ∃φ^i ∈ Φ_c : (φ^{*_q} ⇒ φ^i) then
16:                     continue
17:                 Φ_c[φ^{*_q}] = Φ_q ∪ {φ^k}
18:                 if a_g ∉ ℐ(φ^{*_q}) then
19:                     Φ_* ← Φ_* ∪ {φ^{*_q}}
20:                     continue
21:                 ℚ.enqueue(φ^{*_q})
22:     return Φ_*
```

- $\mathbb{Q}$: Queue used to keep track of all implied rules that are not new and not redundant (i.e., that can still lead to the generation of a necessary rule). Initially, this queue contains all rules of Φ_{a_g}.
- Φ_c: Map of implied rules of which each is mapped to its contributors. Initially, this map contains all rules of Φ_{a_g} that are mapped to singletons containing themselves.
- Φ_*: Set containing all necessary rules with generator a_g and any contributing set $\Phi \subseteq \Phi_{a_g}$. This set will be returned upon completion (line 22).

After the initialization phase, the algorithm continues to test combinations of rules for generating necessary rules with generator a_g, until $\mathbb{Q}$ is empty (line 7-21). In each loop, a certain rule φ^q is dequeued from $\mathbb{Q}$ (line 8), and its contributors are stored in Φ_q (line 9). Then, any rule φ^k that did not already contribute to the construction of φ^q (line 10) and contributes to the elimination of a_g with φ^q (line 11-12) is tested for generating implied rules Φ_{*_q} in combination with φ^q (line 13). Each implied rule φ^{*_q} can either be (1) redundant to any of the previously generated (or given) rules, in which case it is ignored (line 15-16), (2) not redundant, but new with generator a_g, in which case it is added to Φ_* (line 18-20), or (3) not redundant and not new, in which case it should be

tested further and is, therefore, added to $\mathbb{Q}$ (line 21). Note that in the last two cases, the algorithm adds φ^{*_q} together with its contributors to Φ_c (line 17).

Example 9: Consider again the σ-rules (or σ^+-rules) defined on the stock schema (cfr. Table 3), and $a_g = PC$ and $\Phi_{a_g} = \{\varphi^1, \varphi^3, \varphi^4, \varphi^5, \varphi^6\}$ passed as input to Algorithm 1. When applying the implication function on combinations of two of the given rules, we get $\varphi^{*_1} \equiv LTP < 0 \wedge CP > 0$ (contributors φ^1 and φ^4) and $\varphi^{*_2} \equiv LTP < 0 \wedge CP = 0$ (contributors φ^1 and φ^6) as new rules with generator PC. Moreover, $\varphi^{*_3} \equiv PC < LTP \wedge CP \leq 0$ (contributors φ^3 and φ^6) and $\varphi^{*_4} \equiv PC > LTP \wedge CP \geq 0$ (contributors φ^4 and φ^6) are implied, but not redundant and not new, and, therefore, added to $\mathbb{Q}$. All other combinations only lead to redundant rules. After this, only the combination of φ^1 and φ^{*_4} (contributors φ^1, φ^4 and φ^6) leads to the generation of a necessary rule $\varphi^{*_5} \equiv LTP < 0 \wedge CP \geq 0$, which dominates both φ^{*_1} and φ^{*_2}.

To end this section, we focus shortly on the performance of the σ^+-rule implication algorithm, which highly depends on the number of implied rules that are added to the queue. Therefore, it is possible to count, in a worst-case scenario, the number of rules in Φ_{*_q} each time $\neg\text{Imp}(\neg\varphi^q \wedge \neg\varphi^k, -)$ is executed, assuming that all rules in Φ_{*_q} can be added to $\mathbb{Q}$. This count can be determined in a similar way as done in the proof of Corollary 1. Although, one should take into account that, each time, only two σ^+-rules (φ^q and φ^k) are used in the implication function at the cost of generating all implied rules (and not only new rules). In the end, we can state that each Φ_{*_q} contains at most

$$3^{|\varphi^q| \cdot |\varphi^k|} \tag{5}$$

σ^+-rules. Now, the theoretical, worst-case complexity of executing the entire σ^+-rule implication algorithm, assuming that all rules are added to $\mathbb{Q}$, is often worse than the theoretical, worst-case complexity of Theorem 1. However, the strength of the algorithm is that a very large share of the generated implied rules that are captured in Φ_{*_q} are either new or redundant, and are not added to the queue for further extension. Indeed, if we consider Example 9 again, we can see that, when only taking into account the number of rules generated by means of two given rules, 1044 implied rules are generated in total, of which 2 are necessary (i.e., φ^{*_1} and φ^{*_2}), and 2 are not redundant and not new, and therefore added to the queue (i.e. φ^{*_3} and φ^{*_4}). Further, 342 implied rules are generated by extending φ^{*_3} or φ^{*_4}, of which only 1 is necessary (i.e., φ^{*_5}) and all others are redundant. After this, the algorithm terminates, such that only 1386 implied rules are tested instead of 10^8 when using Theorem 1. Later, in VI-B, we will analyze the effects of Algorithm 1 compared to Theorem 1 by evaluating it in a more practical setting and give a recommendation on when to use which procedure.

V. DISCUSSION AND RELATED WORK

One of the most important aspects in data quality handling research is the study of methods to resolve data inconsistencies by means of data quality rules in a (semi-) automated way [4], [22], [26], [27]. The problem of resolving data inconsistencies can be tackled as a process with two steps: error detection/localization (i.e., determining the attribute values in error) and error correction (i.e., estimating correct values for the attribute values in error). Because our contributions mainly focus on error localization, we will give an overview of state-of-the-art contributions in this regard and position our work among these contributions in the remainder of this section.

The last decades, many error localization techniques have been proposed (cfr. Table 6). Examples include vertex generation methods [28], [29], branch-and-bound methods [28], [30], [31] and holistic [19], [32] and probabilistic methods [33], [34], [35], [36]. However, one of the most elegant and easy to understand methods to solve the error localization problem is the set cover method, which is proposed by Fellegi & Holt for regular edit rules [12]. Confirmed by the lifting property, this method only works on sufficient sets of rules [12], [15], [16], which can be constructed by means of rule implication, exploiting attribute elimination. For many types of data quality rules that are typically more expressive than edit rules (e.g., (conditional) functional dependencies [6], [7], [8], [9] and denial constraints [11]), the lifting property does not hold [37]. In such cases, it is not possible to solve the error localization problem by means of the set cover method. Because of this, other error detection and repair frameworks, such as Llunatic (based on the Chase algorithm) [18] and HoloClean [19], have been proposed to repair data failing more expressive data quality rules. However, these frameworks may have scalability problems when used in combination with a mix of different types of rules, with a large number of rules or when a large amount of data to repair is passed as input. As a solution, they rely on heuristics or estimates, in order to complete successfully. In this regard, data quality rules that (i) have expressiveness in between edit rules and denial constraints and (ii) come with an efficient rule implication and set cover repair method, provide an appealing trade-off between expressiveness and efficiency. This resulted in the introduction of σ-rules, which are based on the selection operator σ of relational algebra and to which the concepts and properties of edit rules and edit rule implication can easily be transferred.

A similar trade-off in complexity has also been observed for linear edit rules. In [22], it was stated that there are problems with edit rule implication featuring integer-valued data. Indeed, the properties, methods and algorithms for rule implication with nominal data [12], [15], [16] and continuous data [20] are quite straightforward. However, for integer data (or mixed data), linear edit rules encounter the same problems with gaps as stated in III-C and, thereby, it is not guaranteed that an integer solution always exist. Therefore, one has to rely on a complex post-processing step in terms of shadow regions during Fourier-Motzkin elimination in which this is verified [22], [30], [31]. In our work, we overcome this problem by generalizing σ-rules to σ^+-rules (being a

TABLE 6. Positioning of selection rules among state-of-the-art methods for error localization in data restricted by data quality rules.

method type	frameworks/methods
Vertex generation methods	De Waal [28], [29]
Branch-and-bound methods	De Waal [28], [31], De Waal and Quéré [30]
Holistic methods	Holoclean [19], [32]
Probabilistic methods	Krishnan et al. [33], Mahdavi et al. [34], [35], Pham et al. [36]
Set cover methods	Fellegi & Holt [12], **selection rules**

simplification of linear edit rules), which have the option to easily represent gaps between attribute values and for which, as a direct consequence, no verification step is needed because an integer solution always exists (if the rules in the set do not contradict to each other). In other words, rule implication as the basis of solving the error localization problem by means of the set cover method for σ^+-rules is guaranteed to work correctly and completely.

Finally, with regard to the efficiency of rule implication, many optimizations for edit rules were studied and proposed in previous works. These optimizations are related to the properties of edit rules that can help in applying rule implication more efficiently. The main reason for this is that the initial implication procedure proposed by Fellegi & Holt can become very labour intensive when many edit rules are passed as input. Contributions to this are due to Liepins [38], [39], [40], Garfinkel et al. [15], [20], Winkler [41], [42], Boskovitz [16], Chen [43], [44] and Boeckling [17]. With this in mind, our final contribution in this paper is the study of these optimizations adapted to the setting of σ^+-rules and the proposal of an efficient σ^+-rule implication algorithm exploiting these optimizations.

VI. EVALUATION

In the following, we evaluate the scalability and the error detection/correction ability of using selection rules (σ^+-rules in particular) over a set of experiments. More specifically, we try to provide an answer to the following questions.

- What is the scalability of repairing data with selection rules?
- What is the impact of the proposed optimizations on the σ^+-rule implication algorithm?
- What is the ability of selection rules to detect and correct erroneous data?
- What is the impact of repairing data with selection rules on the amount of erroneous data?

All experiments are executed on a machine running Ubuntu 21.04, with an Intel Core i9-10920X CPU (3.50 GHz, 12 cores) and 64 GB of RAM.

A. EXPERIMENTAL SETUP

In order to answer the questions stated above, we consider six different approaches to evaluate on four different datasets. Details about the setup of the experiments are given below.

1) APPROACHES

In the experiments, we use three different repair strategies for σ^+-rules and compare the results of these strategies with three different configurations of HoloClean [19]. The reason to choose HoloClean to compare with is because it is, to the best of our knowledge, one of the best repair engines in terms of error detection/correction based on data quality rules. Other well-performing, state-of-the-art repair engines, such as Raha/Baran [34], [35] and Spade [36], are not based on data quality rules and are, therefore, not taken into account.

a: σ^+-RULE-BASED REPAIR STRATEGIES

For evaluating σ^+-rules, we consider three constant-cost repair strategies. By this, we mean that the cost to change a value v into v' is a constant, positive integer, fixed per attribute, and, therefore, does not depend on v or v'. Although this is a very simple approach and it is not so flexible as non-constant cost strategies, which have the ability to account for a rich variety of repair strategies, potentially taking into account certain error mechanisms (e.g., single digit errors, rounding errors, phonetic errors,...), the advantage of constant-cost repair strategies is that the set cover method can be applied to find minimal-cost repairs (cfr. III-B). This makes constant-cost strategies easy to understand and quite fast compared to non-constant cost strategies. The three repair strategies for σ^+ rules are listed below. Note that they all three are closely related to the strategies for regular edit rules, introduced by Fellegi & Holt [12].

- **S**equential repair (σ^+-rules (S)): With this strategy, one minimal solution is picked at random out of all potential minimal solutions. Attributes in this solution are repaired one at a time in random order. This is done by choosing, for each attribute, one repair value from the set of permitted values, which is created based on the values of the attributes that do not need a repair and the values of the attributes that are already repaired.
- **J**oint repair (σ^+-rules (J)): Again, a minimal solution is picked at random. The difference with sequential repair is that all attributes in the solution are repaired *jointly*, in order to preserve joint distributions. This is done by picking a donor-tuple among the consistent tuples for which the values of the attributes to repair are covered by the set of permitted values of these attributes. The values of the attributes to repair are then copied from the donor-tuple to the inconsistent tuple. One problem with this strategy is that, sometimes, no donor-tuple is found. If this is the case, we will fall back on sequential repair.
- **C**onditional **s**equential repair (σ^+-rules (CS)): This strategy is an extension of the sequential repair strategy, but chooses a minimal solution in a more intelligent,

less random way, which is based on the lift (a notion of correlation) that would be observed if the values of some attributes are kept. The reason for this is that random selection of a (minimal) solution has been criticized, because it can lead to repairs that would be very infrequent and thereby inflate the frequency of such improbable value combinations [22].

Note that we did not yet elaborate on strategies to pick one particular repair value (in case of the sequential repair approaches) or donor-tuple (in case of joint repair) if multiple possibilities exist. The reason for this is that this might depend on the characteristics of the relation schemata or the datasets. More information on this matter is given in VI-D. Moreover, we used custom implementations of all concepts and algorithms related to σ^+-rules in Java 8, of which the source code is provided in the open-source ledc-sigma package.[5]

b: HoloClean CONFIGURATIONS

HoloClean is a framework for holistic data repairing driven by probabilistic inference exploiting different types of data quality (integrity) constraints [19]. It can make use of three different error detection mechanisms, which are the **N**ullDetector (HoloClean (N)), the **V**iolationDetector (HoloClean (V)) and a combination of the **N**ullDetector and the **V**iolationDetector (HoloClean (NV)). In the experiments, we will use three different configurations of HoloClean, each using a different error detection mechanism. For the other parameters, we kept the default values as provided. HoloClean is written in Python 3.6, using a PostgreSQL (version 9.4+) database in the backend. The source code of HoloClean is also provided as open-source software.[6]

2) DATASETS

a: CHARACTERISTICS

The experiments are conducted on four real-world datasets (referred to as Al, Eu, St1, St2) over three schemata: **al**lergen, **eu**dract and **st**ock. Moreover, each dataset comes with a set of correct tuples, which forms the golden standard of the dataset. The subset of tuples in the dataset for which correct tuples are provided in the golden standard is called the overlap. The characteristics of these schemata (name and value type), corresponding datasets (id, number of tuples $|R|$ and number of attributes $|\mathcal{R}|$), golden standards ($|R|$ and $|\mathcal{R}|$) and overlaps ($|R|$ and $|\mathcal{R}|$) are provided in Table 7. For $|\mathcal{R}|$, we distinguish between the number of attributes that is considered for repair and the number of remaining attributes (shown between brackets).

- allergen: The dataset over this schema captures data related to food products and their allergens. Detailed information about the construction of the dataset and golden standard is given in [45]. We use a slightly modified version in our experiments in which we combine, for each food product, the information provided by the Alnatura[7] webshop and the Open Food Facts[8] registry in one tuple.
- eudract: The dataset over this schema captures data related to the design of clinical trials as reported by the EudraCT registry.[9] Detailed information about the construction of the dataset and golden standard is given in [46]. Note that the golden standard does not contain correct values for attributes placebo and active_comparator. Still, we decided to consider these attributes for repair, as they are involved in the set of σ^+-rules and, therefore, may contribute to the repair of other attributes.
- stock: The datasets over this schema capture data related to daily prices of 1000 stocks, reported by 54 sources. Detailed information about (the construction of) the dataset and golden standard can be consulted and downloaded freely on the web.[10] For our experiments, we used a subset of the available data, in which we only considered continuous attributes with constraints (in the form of σ^+-rules) defined on and in which we only considered tuples capturing data of the first week of July 2011. An important remark is that the values of two attributes (change_percentage and change_in_dollar) can be calculated based on (linear) relationships between the attributes previous_close and last_trading_price (cfr. Example 1). We will show later (cfr. VI-D) that these (linear) relationships can be exploited for calculating repair values. In the provided golden standard, tuples that were inconsistent against these relationships were manually corrected, if possible. Otherwise, the tuples were left out. Note, also, that we will use a second dataset over this schema, only consisting of tuples that appear in the overlap for which the date is July 6th, 2011. The reason for this is that HoloClean was unable to execute successfully on the larger dataset, because the machine ran out of memory resources during execution. The smaller dataset was, therefore, used to obtain an estimate of the performance of HoloClean.

b: QUALITY

In Figure 2, an overview is given of the relative number of errors in the overlap of each dataset compared to the golden standard, both on cell-level and on tuple-level. On cell-level, the relative number of `NULL`-values and the relative number of cells of which the values in the overlap differ from the values of the corresponding cells in the golden standard (i.e., diff-values) are shown. On tuple-level, the relative number of tuples with `NULL`-values, the relative number of tuples with diff-values, and the relative number of tuples with either a `NULL`-value or a diff-value (i.e., an error) are shown.

[5] https://gitlab.com/ledc/ledc-sigma
[6] https://github.com/HoloClean
[7] https://www.alnatura.de/de-de/
[8] https://world.openfoodfacts.org/
[9] https://www.clinicaltrialsregister.eu/
[10] https://lunadong.com/fusionDataSets.htm

TABLE 7. Characteristics of three schemata with their datasets, golden standards and overlaps. For the number of attributes $|\mathcal{R}|$, we distinguish between the number of attributes that is considered for repair and the number of remaining attributes (shown between brackets).

schema		dataset			golden standard		overlap	
name	value type	id	$\|R\|$	$\|\mathcal{R}\|$	$\|R\|$	$\|\mathcal{R}\|$	$\|R\|$	$\|\mathcal{R}\|$
allergen	ordinal	Al	580	42 (+1)	103	42 (+1)	103	42 (+1)
eudract	nominal	Eu	86670	10 (+3)	594	8 (+1)	3133	8 (+3)
stock	continuous	St1	259820	7 (+3)	500	7 (+2)	26257	7 (+3)
	continuous	St2	5269	7 (+3)	100	7 (+2)	5269	7 (+3)

From Figure 2, we can notice the following. First, the allergen-based dataset (Al) does not contain any NULL-values. This implies that error detection can only rely on the σ^+-rules defined over this schema, which makes it more difficult to correctly localize the errors. Moreover, HoloClean without ViolationDetector is not applicable on Al. Second, Al contains, relatively speaking, the highest number of erroneous tuples, but the lowest number of erroneous cells. This means that, in relative terms, the Al dataset has few errors, but the errors are highly distributed across the tuples. Third, the stock-based datasets (St1 and St2) have the highest number of erroneous cells. Also, the distribution of errors in St1 and St2 is more or less the same and, therefore, we can state that St2 is quite representative for evaluating error detection/correction on the stock schema, although it consists of far fewer tuples than St1.

c: DEFINED σ^+-RULES

In Table 8, an overview is given of the number of σ^+-rules defined over each relation schema. Note that, for each schema, a distinction is made between domain rules (which are always constant, because, by definition, there is always only one attribute involved in) and non-domain rules (both variable and constant). Moreover, to give a notion of the work that needs to be done to generate a sufficient set (by applying the FCF algorithm), a distinction is made between the number of rules in the sufficient set and the number of rules that are initially defined (shown between brackets).

With respect to the information provided in Table 8, we can say the following. First, for the allergen schema and the eudract schema, a domain rule is defined for each attribute that is considered for repair. This is not the case for the stock schema, because attributes change_percentage and change_in_dollar are allowed to take any value in their domain. Regarding the non-domain rules, we can see that only variable σ^+-rules are defined over the allergen schema and only constant σ^+-rules are defined over the eudract schema. The stock schema features a combination of constant and variable σ^+-rules. Moreover, note that, for the eudract schema, the number of σ^+-rules in the sufficient set equals the number of σ^+-rules in the initially given set, which is not the case for the other schemata. This is because the sufficient set defined over the eudract schema equals the initially given set.

B. SCALABILITY

In a first set of experiments, we evaluate the scalability of repairing with σ^+-rule-based approaches compared to repairing with HoloClean. First, we measure, for each approach and dataset, the repair execution time. Second, we evaluate the effectiveness of the σ^+-rule implication algorithm compared to applying Theorem 1.

TABLE 8. Overview of the number of σ^+-rules in the sufficient set and the number of initially given σ^+-rules defined over each schema (shown between brackets). A distinction is made between domain rules, variable and constant non-domain rules.

schema	domain	non-domain		total
		variable	constant	
allergen	42 (42)	49 (28)	0 (0)	91 (70)
eudract	10 (10)	0 (0)	9 (9)	19 (19)
stock	5 (5)	20 (9)	9 (4)	25 (14)

The execution times that the different approaches need to repair the datasets entirely are listed in Table 9. Note that, for the HoloClean-based approaches, we distinguish between error detection and error correction and, for the σ^+-rule-based approaches, we distinguish between sufficient set generation with FCF and repair. First, the execution times for error detection and correction in case of HoloClean, and repair in case of σ^+-rules are directly proportionate to the size of the dataset. Moreover, the number of attributes, initially given rules and rules in the generated sufficient set impacts the execution time of the FCF algorithm, as expected. Second, we can state that, in general, HoloClean takes (much) more time to execute, especially if the time to generate a sufficient set is limited and the size of the dataset tends to increase. We already stated earlier that, in this regard, we did not manage to execute HoloClean successfully on the St1 dataset, because the machine ran out of memory resources. Only for the Al dataset, HoloClean performs better, which is due to the fact that generating a sufficient set takes quite long, because of the high number of attributes, rules and implicit relations between these rules and because of the very small number of rows. Although this is the case, note that generating a sufficient set should typically be done only once and it can be reused afterwards each time that one wants to repair a dataset over this schema with any of our approaches. This is because the set of rules depends on the schema and not on the datasets. On the other hand, in an ideal case, HoloClean should be executed entirely every time a dataset changes, because it exploits the value distributions in these datasets for training. Third, if we only consider the σ^+-rule-based approaches, joint repair needs, generally speaking, most time to execute (but has the better error correction ability, cfr. VI-D), whereas (conditional) sequential repair is (much) faster.

To test the effectivity of the properties exploited by the σ^+-rule implication algorithm compared to the procedure

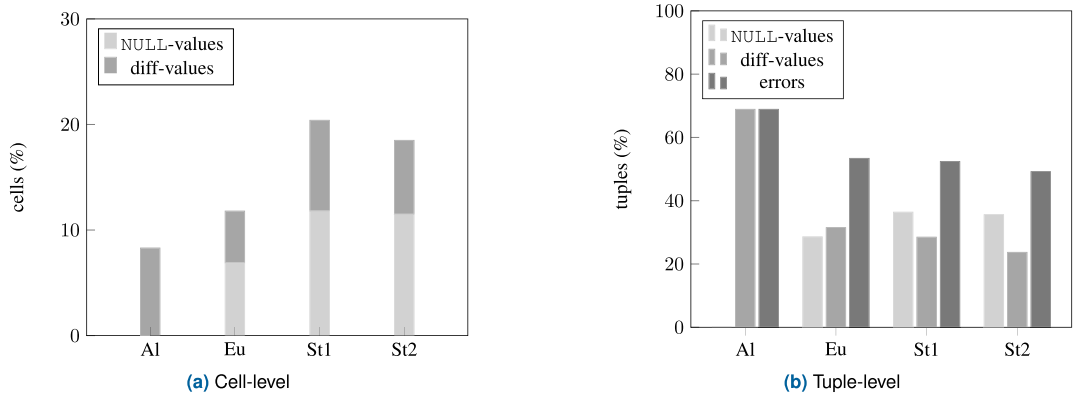

FIGURE 2. Overview of the relative number of errors in the overlap of each dataset compared to the golden standard, both on cell-level and on tuple-level. A distinction is made between NULL-values and cells of which the values differ from the values of the corresponding cells in the golden standard (i.e., diff-values).

TABLE 9. Overview of the repair execution times (in seconds). For the HoloClean-based approaches, we distinguish between error detection and error correction and, for the σ^+-rule-based approaches, we distinguish between sufficient set generation with FCF and repair. N/A indicates that the approach is not applicable. N/E indicates that the approach could not be executed successfully due to limitations in resources.

	Al			Eu			St1			St2		
approach	**detection**	**correction**	**total**	**detection**	**correction**	**total**	**detection**	**correction**	**total**	**detection**	**correction**	**total**
HoloClean (NV)	0.81	58.85	59.66	6.73	2303.37	2310.1				1.11	183.47	184.59
HoloClean (N)	N/A			5.11	2379.69	2384.8	N/E			0.27	157.82	158.09
HoloClean (V)	0.76	58.59	**59.35**	1.6	2083.62	2085.22				0.95	162.4	163.35
	FCF	**repair**	**total**	**FCF**	**repair**	**total**	**FCF**	**repair**	**total**	**FCF**	**repair**	**total**
σ^+-rules (S)	328.1	0.83	328.93	0.01	15.41	**15.42**	20.75	719.1	**739.85**	20.75	2.04	**22.79**
σ^+-rules (J)		1.67	329.77		50.32	50.33		2235.18	2255.93		2.86	23.61
σ^+-rules (CS)		2.41	330.51		21.88	21.89		858.5	879.25		2.22	22.97

proposed in Theorem 1, we count the number of implied rules that will be generated by both methods when passing a given generator as input. For the stock schema, we already stated in IV-C2 that the σ^+-rule implication algorithm has a large advantage over Theorem 1. Especially for the attributes involved in many σ^+-rules (e.g., previous_close, change_percentage,...), the number of generated implied rules is highly reduced. When considering the eudract schema, only attributes open, single_blind and double_blind can lead to necessary rules, because the other attributes cannot be eliminated. Each of these attributes is involved in four σ^+-rules, of which one is featuring one predicate, two are featuring two predicates and one is featuring three predicates, resulting in an upper bound of $6^{12} \approx 2 \cdot 10^9$ new rules (see (4)). When executing Algorithm 1 with each of these attributes as input, merely around 5000 implied rules are generated (and tested). Only for the allergen schema, Theorem 1 is better performing in terms of number of generated rules. The main reason for this is that all σ^+-rules feature only one predicate, resulting in an exponent in (4) that is always equal to 1, and few candidate contributors per attribute exist, resulting in a low base. Therefore, we can conclude that the choice of procedure highly depends on the number of candidate contributors and the number of predicates involved in the candidate contributors. As a general rule of thumb, we recommend to use Algorithm 1 when the number of candidate contributors is larger than 2 (resulting in a base equal to at least 3) and the average number of predicates per candidate contributor is at least 2.

C. ERROR DETECTION

In a second set of experiments, we will evaluate the error detection ability of σ^+-rules compared to the HoloClean-based approaches. More specifically, we will evaluate how well the different approaches perform in detecting erroneous tuples and erroneous cells in the overlap of each of the datasets. The reason to distinguish between error detection and error correction is that, if an approach performs well in detecting errors but performs poorly in correcting errors, we can still rely on other approaches (e.g., domain experts, deductive (i.e., certain) repairs,...) to find correct values.

1) DETECTING ERRONEOUS TUPLES

First, we evaluate the ability to detect erroneous tuples in the overlap of each of the four datasets. In other words, we evaluate how well the different approaches perform in finding tuples in which they are going to change at least one value (i.e., which they consider as erroneous). For this, we report on the following evaluation metrics for each dataset and approach.

- Precision (P): the number of tuples correctly identified as erroneous divided by the number of tuples identified as erroneous.
- Recall (R): the number of tuples correctly identified as erroneous divided by the number of erroneous tuples.
- F_1-score (F_1): harmonic mean of precision and recall.

In Table 10, an overview of the results is given. Note that we did not make a distinction between the σ^+-rule-based

TABLE 10. Overview of the results regarding the ability to detect erroneous tuples. N/A indicates that the approach is not applicable. N/E indicates that the approach could not be executed successfully due to limitations in resources.

	Al			Eu			St1			St2		
approach	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
HoloClean (NV)	**1**	0.31	0.47	**1**	0.37	0.54				**1**	0.78	0.87
HoloClean (N)	N/A			**1**	0.29	0.45	N/E			**1**	0.72	0.83
HoloClean (V)	**1**	0.31	0.47	**1**	0.35	0.52				**1**	0.61	0.76
σ^+-rules (All)	**1**	**0.89**	**0.94**	**1**	**0.62**	**0.76**	**1**	**0.77**	**0.87**	**1**	**0.82**	**0.9**

approaches. The reason for this is that error detection with σ^+-rules only depends on the expressiveness of the σ^+-rules and does not depend on the repair strategy (cfr. VI-A1). Indeed, each tuple failing any of the σ^+-rules or containing a NULL-value is detected as erroneous. Besides that, the precision is 1 for each approach and dataset. If this was not the case, tuples would exist in the golden standard that fail any of the σ^+-rules or that contain NULL-values. For the recall and F_1-score, the σ^+-rule-based approaches always outperform the HoloClean-based approaches, especially on the Al and Eu datasets.

2) DETECTING ERRONEOUS CELLS

Second, we evaluate the ability to detect erroneous cells in the overlap of each of the four datasets. In other words, we evaluate how well the different approaches perform in finding cells which value they are going to change (i.e., which they consider as erroneous). Note that, for the σ^+-rule-based approaches (i.e., the constant-cost repair strategies), this depends on the cost assigned to each attribute to change its value (i.e., the cost model). In order to give a notion of the error detection ability of these approaches and for simplicity reasons, we only consider one cost model for each schema. For the allergen and eudract schema, an equal cost is assigned to each attribute. For the stock schema, a lower cost is assigned to these attributes which value can potentially be determined in a deductive way, because of the (linear) relationships that exist between these attributes, or which contain derived data. More specifically, we assigned cost 1 to change_percentage and change_in_dollar, cost 2 to previous_close and last_trading_price and cost 3 to the other attributes. We report on the following evaluation metrics for each dataset and approach.

- Precision (P): the number of cells correctly identified as erroneous divided by the number of cells identified as erroneous.
- Recall (R): the number of cells correctly identified as erroneous divided by the number of erroneous cells.
- F_1-score (F_1): harmonic mean of precision and recall.

In Table 11, an overview of the results is given. Note that we did not make a distinction between the sequential and joint repair strategy, because of the fact that these approaches search for attributes to repair in the same way (cfr. VI-A1). First, a straightforward insight is that, when errors are due to NULL-values rather than due to erroneous values, error detection is more easy, because cells with NULL-values will always be identified correctly as erroneous. Second, although the σ^+-rule-based approaches score best for detecting erroneous tuples in the Al dataset, they score worst in detecting erroneous cells in this dataset compared to applying the σ^+-rule-based approaches on the other datasets. This is because of the high number of attributes and σ^+-rules, the lack of NULL-values, and the relative high number of inconsistent tuples. Indeed, we noticed that, for each inconsistent tuple, the approaches can choose between a high number of minimal solutions (in terms of attributes to repair) for this dataset. Although this is the case, we can safely state that, for all datasets, the σ^+-rule-based approaches (especially the σ^+-rules (CS) approach) generally outperform the HoloClean-based approaches. Only for the Eu dataset, the HoloClean (N) approach is slightly better in terms of precision. For the Al dataset, σ^+-rules (S/J) slightly outperforms σ^+-rules (CS). Also, note that the results shown in Table 11 serve as upper bounds for the error correction results discussed below.

D. ERROR CORRECTION

In a third and final set of experiments, we evaluate the error correction ability of σ^+-rules compared to the HoloClean-based approaches. First, we evaluate how well the different approaches perform in finding correct values for erroneous cells in the overlap of each of the four datasets. Second, we check how many erroneous tuples still exist after finishing the repair process.

Before discussing the results, we elaborate on which selection strategy has been used to pick a particular repair value (in case of the sequential repair approaches) or donor-tuple (in case of joint repair) if multiple possibilities exist. For the Al and Eu datasets, the (conditional) sequential repair approach picks a value in a frequency-driven way. From the consistent tuples, the frequency of each of the permitted values is measured and each of these values can be chosen with a probability proportionate to their observed frequency. For the stock-based datasets, if it is possible to determine a value deductively (i.e., for attributes change_percentage, change_in_dollar, previous_close and last_trading_price) due to the (linear) relationships between them and if this value is permitted, it is chosen. If not, we fall back on the frequency-driven strategy. Regarding joint repair, donor-tuples are, also, picked in a frequency-driven way, with more frequent donors more likely to be chosen. For the Eu, (resp. St1 and St2) datasets, a donor is chosen in a frequency-driven way from the set of consistent tuples that refer to the same clinical trial (resp. stock-date combination), if possible.

TABLE 11. Overview of the results regarding the ability to detect erroneous cells. N/A indicates that the approach is not applicable. N/E indicates that the approach could not be executed successfully due to limitations in resources. For all σ^+-rule-based approaches, the results are shown as averages over 10 executions.

	Al			Eu			St1			St2		
approach	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
HoloClean (NV)	0.48	0.03	0.06	0.88	0.37	0.52				**1**	0.66	0.8
HoloClean (N)	N/A			**1**	0.37	0.54	N/E			**1**	0.62	0.77
HoloClean (V)	0.48	0.03	0.06	0.84	0.26	0.39				**1**	0.35	0.52
σ^+-rules (S/J)	**0.59**	**0.27**	**0.37**	0.93	0.61	0.74	0.97	0.62	0.75	0.96	0.66	0.78
σ^+-rules (CS)	0.5	0.23	0.32	0.97	**0.64**	**0.77**	**1**	**0.64**	**0.78**	**1**	**0.69**	**0.82**

TABLE 12. Overview of the results regarding the ability to correct erroneous cells. N/A indicates that the approach is not applicable. N/E indicates that the approach could not be executed successfully due to limitations in resources. For all σ^+-rule-based approaches, the results are shown as averages over 10 executions.

	Al			Eu			St1			St2		
approach	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
HoloClean (NV)	0.08	0.01	0.01	0.52	0.22	0.31				**1**	**0.66**	**0.8**
HoloClean (N)	N/A			0.59	0.22	0.32	N/E			**1**	0.62	0.77
HoloClean (V)	0.08	0.01	0.01	0.58	0.17	0.26				**1**	0.35	0.52
σ^+-rules (S)	0.42	0.2	0.27	0.63	0.42	0.5	0.37	0.24	0.29	0.42	0.29	0.34
σ^+-rules (J)	**0.44**	**0.21**	**0.28**	**0.8**	**0.53**	**0.64**	**0.85**	**0.54**	**0.66**	0.89	0.6	0.71
σ^+-rules (CS)	0.35	0.16	0.22	0.67	0.44	0.53	0.4	0.26	0.32	0.47	0.32	0.38

TABLE 13. Overview of the percentage of erroneous tuples in the repaired datasets compared to the original datasets. N/A indicates that the approach is not applicable. N/E indicates that the approach could not be executed successfully due to limitations in resources.

	Al			Eu			St1			St2		
approach	NULL-values	rule errors	errors	NULL-values	rule errors	errors	NULL-values	rule errors	errors	NULL-values	rule errors	errors
Original	0	60.5	60.5	51	11.1	54.4	40.3	7.1	44.4	39	6.3	43.3
HoloClean (NV)	0	58.4	58.4	42.6	6.3	44.9				0	4.9	4.9
HoloClean (N)	N/A			42.6	14.3	47.9	N/E			0	8.2	8.2
HoloClean (V)	0	58.4	58.4	43.3	4.2	45.2				21.2	4.8	25.5
σ^+-rules (All)	0	0	0	0	0	0	0	0	0	0	0	0

The results regarding the ability to correct erroneous tuples are listed in Table 12 and we report on the following metrics.

- Precision (P): the number of correctly repaired cells divided by the number of repaired cells.
- Recall (R): the number of correctly repaired cells divided by the number of erroneous cells.
- F_1-score (F_1): harmonic mean of precision and recall.

We can state that the results are more or less in line with the results regarding error detection on cell-level. For the Al and Eu datasets, the σ^+-rule-based approaches outperform HoloClean for all metrics and approaches. Only for the St2 dataset, HoloClean is better. The reason for this is that there are many (up to 54) sources reporting on the same stock and HoloClean is able to learn repairs based on correlations between attributes (values). For the St1 dataset, we cannot compare with HoloClean, because it did not execute successfully due to limited memory resources (cfr. VI-A2). If we only consider the σ^+-rule-based approaches, we can see that σ^+-rules (J) is, generally speaking, the better one. Especially for the Eu, St1 and St2 datasets, the difference with the other approaches is quite big, because of the ability to use an optimized donor selection strategy, as stated earlier. The difference in performance between the sequential and conditional sequential repair strategy is negligible and depends on the dataset (with a small overall advantage for σ^+-rules (CS)). Based on these findings, we can conclude that the performance of error correction highly depends on the performance of error detection and the ability to use optimized value selection strategies, but, also, on the number of permitted values/donors. Indeed, take, for example, a continuous attribute not containing derived data (e.g., today_low in the stock schema). For these kinds of attributes, you may be allowed to choose from a huge set of permitted values and choosing a value from this set is often done quite arbitrarily because a more intelligent strategy does not exist.

To finish this section, we would like to point out that, an important property of all σ^+-rule-based approaches is that they guarantee error-free correction. This, however, is not the case with HoloClean. Indeed, in Table 13, we have listed, for each approach and dataset, what percentage of tuples in the repair still fail any σ^+-rule or contain a `NULL`-value compared to the original dataset. We can see that the overall number of erroneous tuples decreases when applying HoloClean in general (with HoloClean (NV) performing the best), but for the Al and Eu datasets, this is only slightly. Also, for the HoloClean (N/NV) approaches, it is not guaranteed that all original `NULL`-values will contain a non-`NULL`-value after repair (cfr. Eu dataset). Moreover, for the HoloClean (N) approach, the number of errors against σ^+-rules can even increase, which is as expected, because the approach does not take into account any constraints.

VII. CONCLUSION

In this paper, we investigated the concepts and properties of two types of tuple-level constraints, which we called σ- and σ^+-rules (selection rules in short) and which arise from the

selection operator σ of relational algebra. First, σ-rules are an extension of regular edit rules, in the sense that they allow more operators in their representation and they allow variable and constant comparison instead of constant comparison only. Regardless of the increase in expressivity, it was stated that the concepts and properties of edit rules can easily be transferred to the setting of σ-rules. A particularly interesting result is that rule implication with σ-rules, in order to solve the error localization problem by means of the elegant set cover method, can, in most cases, properly be applied. Only when attribute domains are ordinal, we have to rely on a representation that allows gaps between attribute values to keep the number of rules manageable, which is the reason why we proposed σ^+-rules as an extension of σ-rules. In order to assess the usability of selection rules in practical settings, we evaluated the scalability and error detection and correction ability of selection rules compared to HoloClean, which is a state-of-the-art repair engine, over a set of experiments on four real-world datasets. The results show us that our approach outperforms HoloClean in terms of scalability, especially when datasets are very large and the work that FCF has to do is limited. Although, ideally, FCF has to be executed only once per schema, whereas HoloClean is supposed to train its model again after each modification to the data. Moreover, also in terms of error detection and in terms of error correction, our approach outperforms HoloClean, if the number of sources describing an entity in a dataset is limited.

Still, some research questions remain to study in future work. First, although it is possible to capture many errors by means of selection rules, other types of (more expressive) data quality rules (e.g., functional dependencies) exist, which also have proven their utility in the past. Therefore, one might wonder to what extent (implication with) selection rules can account for more expressive constraints without highly increasing the complexity of the repair engine. Besides that, one can also investigate the benefit of using selection rules and other types of constraints jointly. Second, as is the case for many types of data quality rules, an efficient strategy to automatically discover selection rules can be useful. A potential solution to this could be to study and simplify the strategy to discover (tuple-level) denial constraints. Third, instead of restating the properties of edit rules that help in optimizing rule implication to the setting of σ-rules, one can study additional properties (e.g., of variable rules in particular or related to rule folding) that help to reduce the number of combinations to test. Fourth, although the implication-based method with selection rules proofs to perform quite well in terms of error detection, there might exist better (perhaps more complex) strategies for error correction. One potential way of research can, therefore, focus on error correction mechanisms for selection rules depending on different schema (or dataset) characteristics.

APPENDIX. PROOFS

Proof: [Proof of Theorem 1] First, as we are interested in generating new rules with generator a_g, consider a set of σ^+-rules $\Phi_{a_g} = \{\varphi^1, \ldots, \varphi^n\}$, defined over $\mathcal{R} = \{a_1, \ldots, a_k\}$, in which $a_g \in \mathcal{R}$ is involved (cfr. Definition 6). Second, consider a propositional formula, written as $\varphi^* = \varphi^{*_1} \vee \ldots \vee \varphi^{*_m}$ in DNF, defined over $\mathcal{R}$, in which a_g is not involved and for which $\varphi^* \Rightarrow \varphi^1 \vee \ldots \vee \varphi^n$. From Definition 5 and 6, we can state that φ^* is a propositional formula that is implied and new (with generator a_g), and can be generated by

$$\begin{aligned}\varphi^* &\Rightarrow \varphi^1 \vee \ldots \vee \varphi^n \\ &\equiv \neg(\varphi^1 \vee \ldots \vee \varphi^n) \Rightarrow \neg\varphi^* \\ &\equiv \neg\varphi^1 \wedge \ldots \wedge \neg\varphi^n \Rightarrow \neg\varphi^*\end{aligned}$$

from which it follows that

$$\varphi^* \equiv \neg \mathrm{Imp}(\varphi, a_g),$$

with $\varphi = \neg\varphi^1 \wedge \ldots \wedge \neg\varphi^n$. Now, because $\varphi^* = \varphi^{*_1} \vee \ldots \vee \varphi^{*_m}$, a_g is not involved in any φ^{*_i} and φ^* captures all φ^{*_i} such that $\varphi \Rightarrow \neg\varphi^*$ (cfr. Definition 8), we can conclude that φ^* captures all new σ^+-rules φ^{*_i} that can be generated with any $\Phi_c \subseteq \Phi_{a_g}$ as contributing set and a_g as generator. □

Proof: [Proof of Corollary 1] Consider a set of σ^+-rules $\Phi_{a_g} = \{\varphi^1, \ldots, \varphi^n\}$, defined over $\mathcal{R} = \{a_1, \ldots, a_k\}$, in which $a_g \in \mathcal{R}$ is involved. Theorem 1 states that, with these considerations, (3) is applied with $\varphi = \neg\varphi^1 \wedge \ldots \wedge \neg\varphi^n$ (φ can be considered as a propositional formula in conjunctive normal form (CNF)) and attribute a_g as input parameters. First, in order to find the number of (hidden) predicates not involving a_g and implied by φ, φ should be rewritten in DNF by applying the distributive law of Boolean algebra on $\varphi = \neg\varphi^1 \wedge \ldots \wedge \neg\varphi^n$. From this, we can state that the DNF of φ consists of at most

$$m = |\varphi^1| \cdot \ldots \cdot |\varphi^n|$$

conjunctive clauses with, at most, n predicates each. In this formula, $|\varphi^i|$ denotes the number of predicates in φ^i. Now, from IV-A, we know that each of the m conjunctive clauses in the DNF of φ results in at most $\binom{n}{2}$ (hidden) predicates not involving a_g when applying Definition 8. Applying the final negation in (3) results then in a propositional formula with, in CNF, at most m disjunctive clauses consisting of at most $\binom{n}{2}$ predicates. Finally, in order to find all new rules, the result of (3) should again be rewritten in DNF by applying the distributive law of Boolean algebra, resulting in at most

$$\binom{n}{2}^{|\varphi^1| \cdot \ldots \cdot |\varphi^n|}$$

conjunctive clauses not involving a_g (i.e., new σ^+-rules with generator a_g). □

Proof: [Proof of Proposition 1] Consider

- two σ^+ rules φ^r and φ^d, defined over $\mathcal{R}$, for which $\varphi^r \Rightarrow \varphi^d$, and
- two contributing sets $\Phi_r = \{\varphi^r, \varphi\}$ and $\Phi_d = \{\varphi^d, \varphi\}$, with φ any σ^+-rule, defined over $\mathcal{R}$, which lead to the generation of resp. φ^{*_r} and φ^{*_d}.

From Definition 5 and Theorem 1, we know that

$$\varphi^{*_d} \Rightarrow \varphi^d \vee \varphi$$

such that

$$\neg\text{Imp}(\neg\varphi^d \wedge \neg\varphi, -) \equiv \varphi^{*_d}$$

and

$$\varphi^{*_r} \Rightarrow \varphi^r \vee \varphi$$

such that

$$\neg\text{Imp}(\neg\varphi^r \wedge \neg\varphi, -) \equiv \varphi^{*_r}.$$

Moreover, because $\varphi^r \Rightarrow \varphi^d$ is true by assumption,

$$\begin{aligned}
\varphi^{*_r} &\Rightarrow \varphi^r \vee \varphi \\
&\equiv \neg\varphi^{*_r} \vee \varphi^r \vee \varphi \\
&\equiv \neg\varphi^{*_r} \vee (\varphi^r \wedge (\varphi^r \Rightarrow \varphi^d)) \vee \varphi \\
&\equiv \neg\varphi^{*_r} \vee (\varphi^r \wedge (\neg\varphi^r \vee \varphi^d)) \vee \varphi \\
&\equiv \neg\varphi^{*_r} \vee ((\varphi^r \wedge \neg\varphi^r) \vee (\varphi^r \wedge \varphi^d)) \vee \varphi \\
&\equiv \neg\varphi^{*_r} \vee (\varphi^r \wedge \varphi^d) \vee \varphi \\
&\equiv \varphi^{*_r} \Rightarrow (\varphi^r \wedge \varphi^d) \vee \varphi
\end{aligned}$$

and, because of Theorem 1, we can say that

$$\begin{aligned}
\varphi^{*_r} &\equiv \neg\text{Imp}(\neg(\varphi^r \wedge \varphi^d) \wedge \neg\varphi, -) \\
&\equiv \neg\text{Imp}((\neg\varphi^r \vee \neg\varphi^d) \wedge \neg\varphi, -) \\
&\equiv \neg\text{Imp}((\neg\varphi^r \wedge \neg\varphi) \vee (\neg\varphi^d \wedge \neg\varphi), -) \\
&\equiv \neg(\neg\varphi^{*_r} \vee \neg\varphi^{*_d}) \\
&\equiv \varphi^{*_r} \wedge \varphi^{*_d} \\
&\Rightarrow \varphi^{*_r} \wedge \varphi^{*_d}
\end{aligned}$$

From this, it follows that

$$\begin{aligned}
\varphi^{*_r} &\Rightarrow \varphi^{*_r} \wedge \varphi^{*_d} \\
&\equiv \neg\varphi^{*_r} \vee (\varphi^{*_r} \wedge \varphi^{*_d}) \\
&\equiv (\neg\varphi^{*_r} \vee \varphi^{*_r}) \wedge (\neg\varphi^{*_r} \vee \varphi^{*_d}) \\
&\equiv (\neg\varphi^{*_r} \vee \varphi^{*_d}) \\
&\equiv \varphi^{*_r} \Rightarrow \varphi^{*_d}
\end{aligned}$$

□

Proof: [Proof of Proposition 2] Consider two σ^+-rules φ^1 and φ^2, defined over $\mathcal{R} = \{a_1, \ldots, a_k\}$, for which $\varphi^1_{a_g} \Rightarrow \varphi^2_{a_g}$ (with $a_g \in \mathcal{R}$). Now, we can say that

$$\begin{aligned}
&\neg\text{Imp}(\neg\varphi^1 \wedge \neg\varphi^2, -) \\
&\quad\equiv \neg\text{Imp}(\neg(\varphi^1_{a_g} \wedge \varphi^1_{\bar{a}_g}) \wedge \neg(\varphi^2_{a_g} \wedge \varphi^2_{\bar{a}_g}), -) \\
&\quad\equiv \neg\text{Imp}((\neg\varphi^1_{a_g} \vee \neg\varphi^1_{\bar{a}_g}) \wedge (\neg\varphi^2_{a_g} \vee \neg\varphi^2_{\bar{a}_g}), -) \\
&\quad\equiv \neg\text{Imp}((\neg\varphi^1_{a_g} \wedge \neg\varphi^2_{a_g}) \vee (\neg\varphi^1_{a_g} \wedge \neg\varphi^2_{\bar{a}_g}) \\
&\qquad\vee(\neg\varphi^1_{\bar{a}_g} \wedge \neg\varphi^2_{a_g}) \vee (\neg\varphi^1_{\bar{a}_g} \wedge \neg\varphi^2_{\bar{a}_g}), -)
\end{aligned}$$

Because $\varphi^1_{a_g} \Rightarrow \varphi^2_{a_g}$, $\neg\varphi^1_{a_g} \wedge \neg\varphi^2_{a_g} \equiv \neg\varphi^2_{a_g}$, such that we can rewrite this as

$$\begin{aligned}
&\neg\text{Imp}(\neg\varphi^2_{a_g} \vee (\neg\varphi^1_{a_g} \wedge \neg\varphi^2_{\bar{a}_g}) \\
&\qquad\vee(\neg\varphi^1_{\bar{a}_g} \wedge \neg\varphi^2_{a_g}) \vee (\neg\varphi^1_{\bar{a}_g} \wedge \neg\varphi^2_{\bar{a}_g}), -)
\end{aligned}$$

Besides that, we can assume that $\varphi^1_{a_g}$ and $\varphi^2_{\bar{a}_g}$ (resp. $\varphi^1_{\bar{a}_g}$ and $\varphi^2_{a_g}$) have no attributes in common, because if they do, $\varphi^1_{a_g} \not\Rightarrow \varphi^2_{a_g}$. Therefore, $\neg\text{Imp}(\neg\varphi^1_{a_g} \wedge \neg\varphi^2_{\bar{a}_g}, -) \equiv \varphi^1_{a_g} \vee \varphi^2_{\bar{a}_g}$ (resp. $\neg\text{Imp}(\neg\varphi^1_{\bar{a}_g} \wedge \neg\varphi^2_{a_g}, -) \equiv \varphi^1_{\bar{a}_g} \vee \varphi^2_{a_g}$), such that we can rewrite this as

$$\begin{aligned}
&\varphi^2_{a_g} \wedge (\varphi^1_{a_g} \vee \varphi^2_{\bar{a}_g}) \wedge (\varphi^1_{\bar{a}_g} \vee \varphi^2_{a_g}) \\
&\qquad\wedge\neg\text{Imp}(\neg\varphi^1_{\bar{a}_g} \wedge \neg\varphi^2_{\bar{a}_g}, -) \\
&\quad\equiv ((\varphi^2_{a_g} \wedge \varphi^1_{a_g} \wedge \varphi^1_{\bar{a}_g}) \vee (\varphi^2_{a_g} \wedge \varphi^1_{a_g}) \\
&\qquad\vee(\varphi^2_{a_g} \wedge \varphi^2_{\bar{a}_g} \wedge \varphi^1_{\bar{a}_g}) \vee (\varphi^2_{a_g} \wedge \varphi^2_{\bar{a}_g})) \\
&\qquad\wedge\neg\text{Imp}(\neg\varphi^1_{\bar{a}_g} \wedge \neg\varphi^2_{\bar{a}_g}, -)
\end{aligned}$$

Again, because $\varphi^1_{a_g} \Rightarrow \varphi^2_{a_g}$, this is equivalent to

$$\begin{aligned}
&((\varphi^1_{a_g} \wedge \varphi^1_{\bar{a}_g}) \vee \varphi^1_{a_g} \vee (\varphi^2_{a_g} \wedge \varphi^2_{\bar{a}_g} \wedge \varphi^1_{\bar{a}_g}) \\
&\qquad\vee(\varphi^2_{a_g} \wedge \varphi^2_{\bar{a}_g})) \wedge \neg\text{Imp}(\neg\varphi^1_{\bar{a}_g} \wedge \neg\varphi^2_{\bar{a}_g}, -) \\
&\quad\equiv (\varphi^1_{a_g} \vee \varphi^2) \wedge \neg\text{Imp}(\neg\varphi^1_{\bar{a}_g} \wedge \neg\varphi^2_{\bar{a}_g}, -).
\end{aligned}$$

From Theorem 1, we know that $\neg\text{Imp}(\neg\varphi^1_{\bar{a}_g} \wedge \neg\varphi^2_{\bar{a}_g}, -)$ results in σ^+-rules that are redundant to $\varphi^1_{\bar{a}_g}$ or to $\varphi^2_{\bar{a}_g}$, or in σ^+-rules that are implied from $\neg\varphi^1_{\bar{a}_g} \wedge \neg\varphi^2_{\bar{a}_g}$ and do not involve an attribute $a_{g'} \in \mathcal{R}$ other than a_g, because a_g is initially not involved. From this and the initial assumptions, the stated follows automatically. □

REFERENCES

[1] C. T. Redman, *Data Quality for the Information Age*, 1st ed. Norwood, MA, USA: Artech House, 1997.
[2] C. Batini and M. Scannapieco, *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Berlin, Germany: Springer-Verlag, 2006.
[3] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino, "Methodologies for data quality assessment and improvement," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–52, Jul. 2009.
[4] W. Fan and F. Geerts, *Foundations of Data Quality Management*. San Rafael, CA, USA: Morgan & Claypool Publishers, 2012.
[5] A. Bronselaer, R. De Mol, and G. De Tre, "A measure-theoretic foundation for data quality," *IEEE Trans. Fuzzy Syst.*, vol. 26, no. 2, pp. 627–639, Apr. 2017.
[6] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases: The Logical Level*, 1st ed. Reading, MA, USA: Addison-Wesley, 1995.
[7] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2005, pp. 143–154.
[8] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for data cleaning," in *Proc. IEEE Int. Conf. Data Eng.*, Apr. 2007, pp. 746–755.
[9] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *ACM Trans. Database Syst.*, vol. 33, no. 2, pp. 1–48, Jun. 2008.
[10] A. Qahtan, N. Tang, M. Ouzzani, Y. Cao, and M. Stonebraker, "Pattern functional dependencies for data cleaning," *Proc. VLDB Endowment*, vol. 13, no. 5, pp. 684–697, Jan. 2020.
[11] X. Chu, I. F. Ilyas, and P. Papotti, "Discovering denial constraints," *Proc. VLDB Endowment*, vol. 6, no. 13, pp. 1498–1509, Aug. 2013.

[12] I. P. Fellegi and D. Holt, "A systematic approach to automatic edit and imputation," *J. Amer. Stat. Assoc.*, vol. 71, no. 353, pp. 17–35, Mar. 1976.
[13] J. Rammelaere and F. Geerts, "Cleaning data with forbidden itemsets," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 8, pp. 1489–1501, Aug. 2019.
[14] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.
[15] R. S. Garfinkel, A. S. Kunnathur, and G. E. Liepins, "Optimal imputation of erroneous data: Categorical data, general edits," *Oper. Res.*, vol. 34, no. 5, pp. 744–751, Oct. 1986.
[16] A. Boskovitz, "Data editing and logic: The covering set method from the perspective of logic," M.S. thesis, Math. Sci. Inst., Austral. Nat. Univ., Canberra, ACT, Australia, 2008.
[17] T. Boeckling, G. D. Tré, and A. Bronselaer, "Efficient edit rule implication for nominal and ordinal data," *Inf. Sci.*, vol. 590, pp. 179–197, Apr. 2022.
[18] F. Geerts, G. Mecca, P. Papotti, and D. Santoro, "Cleaning data with llunatic," *VLDB J.*, vol. 29, no. 4, pp. 867–892, Jul. 2019.
[19] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "Holoclean: Holistic data repairs with probabilistic inference," *Proc. PVLDB*, vol. 10, no. 11, pp. 1190–1201, Aug. 2017.
[20] R. S. Garfinkel, A. S. Kunnathur, and G. E. Liepins, "Error localization for erroneous data: Continuous data, linear constraints," *SIAM J. Sci. Stat. Comput.*, vol. 9, no. 5, pp. 922–931, Sep. 1988.
[21] L. Bertossi, *Database Repairing and Consistent Query Answering*. San Rafael, CA, USA: Morgan & Claypool Publishers, 2011.
[22] T. D. Waal, J. Pannekoek, and S. Scholtus, *Handbook of Statistical Data Editing and Imputation*. Hoboken, NJ, USA: Wiley, 2011.
[23] J. Fourier, "Solution d'une question particulière du calcul des inégalités," Oeuvres II, Tech. Rep., 1826.
[24] T. Motzkin, "Contributions to the theory of linear inequalities," M.S. thesis, Dept. Math., Univ. Basel, Basel, Switzerland, 1936.
[25] J. R. Duffin, "On fourier's analysis of linear inequality systems," *Math. Program. Stud.*, vol. 1, pp. 71–95, Jan. 1974.
[26] R. M. Groves, F. J. Fowler, M. P. Couper, J. M. Lepkowski, E. Singer, and R. Tourangeau, *Survey Methodology*, 2nd ed. Hoboken, NJ, USA: Wiley, 2009.
[27] I. F. Ilyas, *Data Cleaning*. San Rafael, CA, USA: Morgan & Claypool Publishers, 2019.
[28] T. D. Waal, "Processing of erroneous and unsafe data," M.S. thesis, Dept. Methodol., Erasmus Univ. Rotterdam, Rotterdam, The Netherlands, 2003.
[29] T. D. Waal, "Solving the error localization problem by means of vertex generation," *Surv. Methodol.*, vol. 29, no. 1, pp. 71–80, 2005.
[30] T. D. Waal, "Automatic error localisation for categorical, continuous and integer data," *Statist. Oper. Res. Trans.*, vol. 59, no. 1, pp. 57–99, 2005.
[31] T. D. Waal and R. Quére, "A fast and simple algorithm for automatic editing of mixed data," *J. Off. Statist.*, vol. 19, no. 4, pp. 383–402, 2003.
[32] X. Chu, I. F. Ilyas, and P. Papotti, "Holistic data cleaning: Putting violations into context," in *Proc. IEEE 29th Int. Conf. Data Eng. (ICDE)*, Apr. 2013, pp. 458–469.
[33] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, "Activeclean: Interactive data cleaning for statistical modeling," *Proc. PVLDB*, vol. 9, no. 12, pp. 948–959, Aug. 2016.
[34] M. Mahdavi, Z. Abedjan, R. C. Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Raha: A configuration-free error detection system," in *Proc. Int. Conf. Manage. Data*, Jun. 2019, pp. 865–882.
[35] M. Mahdavi and Z. Abedjan, "Baran: Effective error correction via a unified context representation and transfer learning," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 1948–1961, Jul. 2020.
[36] M. Pham, C. A. Knoblock, M. Chen, B. Vu, and J. Pujara, "SPADE: A semi-supervised probabilistic approach for detecting errors in tables," in *Proc. 30th Int. Joint Conf. Artif. Intell.*, Aug. 2021, pp. 3543–3551.
[37] S. Ginsburg and S. M. Zaiddan, "Properties of functional-dependency families," *J. ACM*, vol. 29, no. 3, pp. 678–698, Jul. 1982.
[38] E. G. Liepins, "Refinements to the Boolean approach to automatic data editing," Oak Ridge Nat. Lab., Oak Ridge, TEN, USA, Tech. Rep. ORNL/TM-7156, 1980.
[39] E. G. Liepins, "A rigorous, systematic approach to automatic data editing and its statistical basis," Oak Ridge Nat. Lab., Oak Ridge, TEN, USA, Tech. Rep. ORNL/TM-7126, 1981.
[40] E. G. Liepins, *Algorithms for Error Localization of Discrete Data*. Oak Ridge, TN, USA: Oak Ridge National Laboratory, 1984.
[41] E. W. Winkler, "Editing discrete data," in *Proc. Surv. Res. Methods Sect.*, 1995, pp. 108–113.
[42] E. W. Winkler, "Set covering and editing discrete data," in *Proc. Surv. Res. Methods Sect.*, 1997, pp. 564–569.
[43] B.-C. Chen, "Set covering algorithms in edit generation," U.S. Bureau Census, Suitland-Silver Hill, MD, USA, Tech. Rep. RR98-06, 1998.
[44] B.-C. Chen and E. W. Winkler, "The cutting plane algorithm in the error localization problem," in *Proc. Joint Stat. Meetings*, 2004.
[45] A. Bronselaer and M. Acosta, "Consistent data fusion with Parker," 2022. [Online]. Available: https://arxiv.org/abs/2202.12184
[46] A. Bronselaer, T. Boeckling, and F. Pattyn, "Dynamic repair of categorical data with edit rules," *Expert Syst. Appl.*, vol. 201, Sep. 2022, Art. no. 117132.

TOON BOECKLING received the B.S. degree in computer science and the M.Sc. degree in computer science engineering from Ghent University, Ghent, Belgium, in 2016 and 2017, respectively. He is currently pursuing the Ph.D. degree with the Research Unit 'Database, Document, and Content Management' under the supervision of Professor A. Bronselaer. His research interests include data quality, data integration, and databases.

GUY DE TRÉ received the M.Sc. degree in computer science and the Ph.D. degree in engineering from Ghent University, Ghent, Belgium, in July 1994 and June 2000, respectively. Since October 2004, he has been a Professor in fuzzy information processing with the Department of Telecommunications and Information Processing, Ghent University, where he heads the Research Unit 'Database, Document, and Content Management.' His research interests include the principles and practice of imperfect information handling in information systems.

ANTOON BRONSELAER received the M.Sc. degree in computer science and the Ph.D. degree in engineering from Ghent University, Ghent, Belgium, in July 2006 and 2010, respectively. Since October 2006, he has been a Researcher with the Department of Telecommunications and Information Processing, Research Unit 'Database, Document, and Content Management,' Ghent University. His research interests include data quality and data integration.