Implication and Axiomatization of Functional Constraints on Patterns with an Application to the RDF Data Model

Jelle Hellings[1], Marc Gyssens[1], Jan Paredaens[2], and Yuqing Wu[3*]

[1] Hasselt University and Transnational University of Limburg
`{jelle.hellings,marc.gyssens}@uhasselt.be`
[2] University of Antwerp
`jan.paredaens@uantwerpen.be`
[3] Indiana University
`yuqwu@cs.indiana.edu`

Abstract. Akhtar et al. introduced equality-generating constraints and functional constraints as an initial step towards dependency-like integrity constraints for RDF data [1]. Here, we focus on functional constraints. The usefulness of functional constraints is not limited to the RDF data model. Therefore, we study the functional constraints in the more general setting of relations with arbitrary arity. We show that a chase algorithm for functional constraints can be normalized to a more specialized *symmetry-preserving chase* algorithm. This symmetry-preserving chase algorithm is subsequently used to construct a sound and complete axiomatization for the functional constraints. This axiomatization is in particular applicable in the RDF data model, solving a major open problem of Akhtar et al.

Keywords: functional constraints, chase algorithm, axiomatization

1 Introduction

Usually, data is subject to integrity constraints implied by the semantics of the data. Formalizing these constraints can help reasoning over the data and help identifying inconsistencies in the data. As such, formal constraints play a major role in database management systems that automatically maintain integrity of the data and optimize query evaluation.

For the relational data model, many types of constraints have been investigated. Among the simplest constraints are the functional dependencies [2]. Functional dependencies play an important role in the well-known Boyce-Codd normal form [3] and in relational schema normalization in general. Besides the functional dependencies, many other dependencies have been investigated (see,

---

e.g., [4, 5]). One of these, the equality-generating dependencies [4], is a natural generalization of the functional dependencies.

For the RDF and XML graph data models, a large body of work on the integrity of data focuses on the schema of the data. Examples are RDF Schema and, for the XML data model, DTDs and XSDs. The usage of dependency-like constraints is less common for these data-models although initial steps have been made (e.g. [1, 6–13]).

An example of dependency-like constraints for the RDF data model are the *equality-generating constraints* and the *functional constraints* of Akhtar et al. [1, 14]. Equality-generating constraints specify patterns that can occur in RDF data, together with equalities that should hold on these patterns. As such, the equality-generating constraints are similar to the equality-generating dependencies of Beeri and Vardi [4], to the equality-generating fragment of the implication dependencies of Fagin [15], and to the full equality-generating dependencies of Wijsen [16]. In these dependencies, the generality of patterns, which allow constants and are untyped, is only matched by the full equality-generating dependencies of Wijsen.

Functional constraints are a generalization of functional dependencies on ternary RDF relations and have the form

$$(P, L \to R),$$

where P specifies a pattern in the RDF data and L and R are sets of variables occurring in this pattern. Their semantics is comparable to that of the functional dependencies: if two parts of the RDF data match the pattern and are equal on L, then they must also be equal on R.

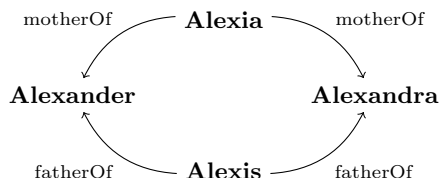*Example 1.* Consider the family tree visualized in Figure 1.

Fig. 1. Simplified visualization of an RDF representation of a small family tree.

On this data, the constraint "a child only has one biological father and mother" holds. This constraint can be expressed by the functional constraints $(\{(\$p, \mathit{fatherOf}, \$c)\}, \$c \to \$p)$ and $(\{(\$p, \mathit{motherOf}, \$c)\}, \$c \to \$p)$. The stronger constraint "children have only one biological parent", which can be expressed by $(\{(\$p, \$t, \$c)\}, \$c \to \$p)$, does not hold on this data.

The functional constraints are subsumed by the equality-generating constraints of Akhtar et al. [1]. Although we shall sometimes refer to equality-generating constraints to describe the general context of this research, the focus

here is on functional constraints. We shall consider functional constraints on relations of arbitrary arity, as the restriction to ternary patterns, as used in the RDF data model, is non-essential.

Functional constraints allow the expression of several types of integrity constraints; these include traditional functional dependencies [2], context-dependent functional dependencies, and constraints on the structure of graphs (described by an edge relation), as illustrated by the following examples.

Example 2. Consider the following relation schema for storing personal information: *PI*(*name*, *ssn*, *address*, *number*, *city*, *postal-code*, *country*), where *ssn* is the social security number. It is natural to add the functional dependency *ssn* → *name* to this scheme. We can express this functional dependency by the functional constraint $(\{(\$na, \$s, \$a, \$nu, \$ci, \$p, \$co)\}, \$s \to \$na)$.

Many integrity constraints are context-dependent. The functional constraints can use patterns and constants in patterns to restrict the context of a standard functional dependency to a subset of the relation.

Example 3. The information represented by postal codes is context-dependent. In the Netherlands, the postal code and house number uniquely identify an address, but this is not the case in Belgium. We thus use a constant for the country to make the functional dependency *postal-code*, *number* → *address*, *city* context-dependent: $(\{(\$na, \$s, \$a, \$nu, \$ci, \$p, \texttt{NL})\}, \$p\$nu \to \$a\$ci)$.

Observe that functional constraints are not the only generalization of the functional dependencies which allow the expression of context-dependent functional dependencies. Other examples include conditional functional dependencies [17] and qualified functional dependencies [18]. The conditional functional dependencies define functional dependencies over a tableau with constants and blanks. The qualified functional dependencies allow the specification of views in which functional dependencies should hold. Patterns are conceptually related to tableaux and to views as tableau queries. Even though functional constraints, conditional functional dependencies, and qualified functional dependencies are related in this way, functional constraints on the one hand and conditional and qualified functional dependencies on the other hand are incomparable, as is argued next.

Example 4. The constraint "Ireland does not have postal codes" can obviously not be expressed as a functional constraint. By using constants in the right-hand side, however, we can express it as the conditional functional dependency $(\{(\$na, \$s, \$a, \$nu, \$ci, \$p, \texttt{IE})\}, \emptyset \to [\$p = \texttt{null}])$.[4]

The use of free variables and constants in patterns cannot be simulated by the tableaux or views used in conditional and qualified functional dependencies, however.

---

[4] We adapted the original notation of conditional functional dependencies to better match our notation of functional constraints.

Because of the use of free variables and constants in patterns, patterns may also match specific structures in the relation. This is particularly useful if the underlying relation represents a graph. In this setting, functional constraints may impose structural constraints.

Example 5. Let $Edge(from, to)$ be a binary relation schema representing the edge relations of a graph. The functional constraint $(\{(\$n, \$n)\}, \emptyset \rightarrow \$n)$ expresses that there is at most one node with a self-loop. The pattern $\{(\$n, \$m), (\$m, \$n)\}$ in the functional constraint $(\{(\$n, \$m), (\$m, \$n)\}, \$n \rightarrow \$m)$ matches cycles (closed paths) of length 2 (including self-loops). Consider two pairs of such cycles starting in node v. By the constraint, the second node in both cycles must be equal, and thus the latter constraint expresses that every node v is part of at most one cycle of length 2.

For the functional dependencies in the relational data model, a sound and complete axiomatization is long known [19]. Akhtar et al. presented a sound and complete axiomatization for the equality-generating constraints in the RDF data model [1]. As functional constraints are subsumed by equality-generating constraints, this axiomatization can also be used for the inference of functional constraints only. In this case, intermediate inference steps can generate equality-generating constraints that are not necessarily equivalent to functional constraints, unfortunately. Akhtar et al. identified the existence of a sound and complete axiomatization of functional constraints (not including other types of constraints) as a major open problem. On the one hand, the Armstrong axiomatization for the functional dependencies [19] can be generalized to the setting of functional constraints. This generalization, however, lacks the reasoning power over patterns necessary for a complete axiomatization. On the other hand, there is no straightforward way to specialize the axiomatization of the equality-generating constraints to functional constraints only.

In this paper, we present a sound and complete axiomatization for the functional constraints over relations of arbitrary arity. In particular, the case of ternary relations yields a sound and complete axiomatization for the functional constraints in the RDF data model, thereby positively solving the open problem of Akhtar et al. [1].

The key insight that led to the breakthrough is that the chase algorithm for equality-generating constraints [1]—which is a variation of the standard chase algorithm [20, 21]—can be normalized to a more specialized, symmetry-preserving, chase algorithm when applied to functional constraints only. The main idea behind the symmetry-preserving chase algorithm is that, due to their semantics, chases for functional constraints always start with tableaux that are symmetric. We prove that during such chases one can always maintain this symmetry in the tableau. Such a symmetry-preserving chase can be described as a sequence of inferences of functional constraints, which in turn leads to the sound and complete axiomatization.

Organization. In Section 2, we present the necessary definitions used throughout this paper. In Section 3, we introduce generalized functional constraints and

equality-generating constraints. In Section 4, the chase algorithm for equality-generating constraints is specialized to functional constraints and subsequently normalized to the symmetry-preserving chase algorithm. In Section 5, we present a sound axiomatization for the functional constraints which suffices to simulate every symmetry-preserving chase, and which is therefore also complete. In Section 6, we conclude on our findings and discuss directions for future work.

2 Preliminaries

Functional and equality-generating constraints [1] have originally been introduced in the context of the RDF data model. In this model, RDF data are usually represented by a single ternary relation. In the Introduction, we have already argued that functional and equality-generating constraints are useful in a wider range of data models. We therefore generalize functional and equality-generating constraints to relations of arbitrary arity. The following notations and definitions will be used throughout the paper.

We consider disjoint infinitely enumerable sets $\mathcal{U}$ and $\mathcal{V}$ of *constants* and *variables*, respectively. For distinction, we usually prefix variables by "$". A *term* is either a constant or a variable. Hence, the set $\mathcal{T}$ of all terms equals $\mathcal{U} \cup \mathcal{V}$. A *tuple* of arity n is a sequence $(t_1, \ldots, t_n)$ of terms. A *pattern* of arity n is a finite set of tuples of arity n. If P is a pattern, then $\mathcal{V}_P$ denotes the set of all variables in P. A *relation* $\mathcal{R}$ of arity n is a pattern of arity n with $\mathcal{V}_\mathcal{R} = \emptyset$.

We define the *domain*, *range*, and *inverse* of a function f in the usual way and denote these by $\mathrm{domain}(f)$, $\mathrm{range}(f)$, and f^{-1}, respectively. Two functions f and g *agree* on a set S, denoted by $f =_S g$, if $f(x) = g(x)$ for all $x \in S$. The *restriction* of a function f to a set S is defined as $f|_S = \{(x, y) \mid x \in S, y = f(x)\}$. The *identity* on a set S is defined as $\mathrm{id}_S = \{(s, s) \mid s \in S\}$. The *extension with identity* of a function f to a set S, $S \cap \mathrm{domain}(f) = \emptyset$, is $f \cup \mathrm{id}_S$.

The *term-based renaming function* $\phi_{a_1 \leftarrow b_1, \ldots, a_i \leftarrow b_i}$, $a_1, b_1, \ldots, a_i, b_i \in \mathcal{T}$, is the function on $\mathcal{T}$ for which $\phi_{a_1 \leftarrow b_1, \ldots, a_i \leftarrow b_i}(b_j) = a_j$, $j = 1, \ldots, i$, and which is the identity elsewhere. Likewise, the *function-based renaming function* $\Phi_{f \leftarrow g}$, with f a function and g an injective function on the same set of variables, is the function on $\mathcal{T}$ for which $\Phi_{f \leftarrow g}(g(\$v)) = f(\$v)$, $\$v \in \mathrm{domain}(g) = \mathrm{domain}(f)$, and which is the identity elsewhere. Notice that this function is well defined due to the injectivity of g.

A function f on terms is extended to tuples, patterns, and sets in the following natural way: for a tuple $(t_1, \ldots, t_n)$, $f((t_1, \ldots, t_n)) = (f(t_1), \ldots, f(t_n))$, and, for a set S, $f(S) = \{f(s) \mid s \in S\}$.

For two patterns P and Q, a function $e : \mathcal{V}_P \cup \mathcal{U} \to \mathcal{T}$ is an embedding of P into Q if $e|_\mathcal{U} = \mathrm{id}_\mathcal{U}$ and $e(P) \subseteq Q$.

We finally review some notation and terminology that can be applied to any type of constraint. "Relation $\mathcal{R}$ *satisfies* constraint C" is denoted by $\mathcal{R} \models C$. A relation $\mathcal{R}$ *satisfies* a set of constraints $\mathcal{C}$, denoted by $\mathcal{R} \models \mathcal{C}$, if, for every $C \in \mathcal{C}$, $\mathcal{R} \models C$. If $\mathcal{C}_1$ and $\mathcal{C}_2$ are sets of constraints then $\mathcal{C}_1$ *implies* $\mathcal{C}_2$, denoted by $\mathcal{C}_1 \models \mathcal{C}_2$, if, for every relation $\mathcal{R}$ with $\mathcal{R} \models \mathcal{C}_1$, we have $\mathcal{R} \models \mathcal{C}_2$. For a set

of constraints $\mathcal{C}$ and a single constraint C, we write $\mathcal{C} \models C$ for $\mathcal{C} \models \{C\}$. The sets of constraints $\mathcal{C}_1$ and $\mathcal{C}_2$ are *equivalent*, denoted by $\mathcal{C}_1 \equiv \mathcal{C}_2$, if $\mathcal{C}_1 \models \mathcal{C}_2$ and $\mathcal{C}_2 \models \mathcal{C}_1$. If, in this notation, $\mathcal{C}_i$, $i = 1$ and/or $i = 2$, is a singleton set $\{C_i\}$, we write C_i for $\mathcal{C}_i$, as before. "Constraint C can be derived from set of constraints $\mathcal{C}$ using the set of *axioms* $\mathfrak{R}$" is denoted by $\mathcal{C} \vdash_{\mathfrak{R}} C$. We usually omit $\mathfrak{R}$ if $\mathfrak{R}$ is clear from the context. The set $\mathfrak{R}$ is *sound* if, for all sets of constraints $\mathcal{C}$ and for all single constraints C, $\mathcal{C} \vdash_{\mathfrak{R}} C$ implies $\mathcal{C} \models C$; it is *complete* if, for all sets of constraints $\mathcal{C}$ and for all single constraints C, $\mathcal{C} \models C$ implies $\mathcal{C} \vdash_{\mathfrak{R}} C$. A set of axioms is an *axiomatization* if it is sound, complete, and recursive.

3 Functional Constraints

We formally define functional constraints on n-ary relations.

Definition 1. *A* functional constraint *is a pair* $(P, L \rightarrow R)$, *where* P *is a nonempty pattern and* $L, R \subseteq \mathcal{V}_P$.

If $C = (P, L \rightarrow R)$ is a functional constraint, then P is the pattern of C, L is the left-hand side of C, and R is the right-hand side of C.

Definition 2. *Let* $\mathcal{R}$ *be a relation and let* $C = (P, L \rightarrow R)$ *be a functional constraint. Then* $\mathcal{R}$ satisfies C *if, for every pair of embeddings* e_1 *and* e_2 *of* P *into* $\mathcal{R}$ *with* $e_1 =_L e_2$, *we have* $e_1 =_R e_2$.

As already mentioned, the functional constraints are a strict subclass of the equality-generating constraints, and the functional constraints are a generalization of the functional dependencies. Below, we formalize these relationships in our setting. This allows us to apply results for equality-generating constraints to functional constraints, and to generalize results for functional dependencies to functional constraints.

3.1 Equality-Generating Constraints

We formally define equality-generating constraints on n-ary relations.

Definition 3. *An* equality-generating constraint *is a pair* (P, E), *where* P *is a nonempty pattern and* E *is a set of equalities of the form* $t_1 = t_2$ *with* $t_1, t_2 \in \mathcal{V}_P \cup \mathcal{U}$.

Definition 4. *Let* $\mathcal{R}$ *be a relation and let* $C = (P, E)$ *be an equality-generating constraint. Then* $\mathcal{R}$ satisfies C *if, for every embedding* e *of* P *into* $\mathcal{R}$ *and every equality* $(t_1 = t_2) \in E$, *we have* $e(t_1) = e(t_2)$.

Akhtar et al. [1] already showed that every functional constraint can be written as an equality-generating constraint. Adopted to our setting, their result is as follows:

Proposition 1. *Let* $C_{\mathrm{FC}} = (P, L \rightarrow R)$ *be a functional constraint. Let* $f_1, f_2 : \mathcal{V}_P \rightarrow \mathcal{V}$ *be injections with* $f_1 =_L f_2$ *and* $\mathrm{range}(f_1|_{\mathcal{V}_P \backslash L}) \cap \mathrm{range}(f_2|_{\mathcal{V}_P \backslash L}) = \emptyset$. *Let* $C_{\mathrm{EGC}} = ((f_1 \cup \mathrm{id}_{\mathcal{U}})(P) \cup (f_2 \cup \mathrm{id}_{\mathcal{U}})(P), \{f_1(\$r) = f_2(\$r) \mid \$r \in R\})$. *Then* $C_{\mathrm{FC}} \equiv C_{\mathrm{EGC}}$.

3.2 Functional Dependencies

We assume familiarity with the functional dependencies of Codd [2, 5].

Proposition 2. *Let $C = L \to R$ be a functional dependency over the relation schema $\mathcal{R} = (A_1, \ldots, A_n)$ with $L, R \subseteq \{A_1, \ldots, A_n\}$. Consider the functional constraint $C_{\text{FC}} = (\{(A_1, \ldots, A_n)\}, L \to R)$, in which the attribute names are assumed to be variables. Then $C \equiv C_{\text{FC}}$.*

The functional dependencies have a well-known axiomatization in the form of *Armstrong's axioms*, consisting of the three axioms *reflexivity*, *augmentation*, and *transitivity* [19]. We generalize Armstrong's axioms to our setting of the functional constraints.

Proposition 3 (Reflexivity). *Let P be a pattern. If $R \subseteq L \subseteq \mathcal{V}_P$, then $(P, L \to R)$.*

Proof (soundness). Let e_1 and e_2 be embeddings of P into a relation $\mathcal{R}$ with $e_1 =_L e_2$. We have $R \subseteq L$ and hence also $e_1 =_R e_2$. □

Proposition 4 (Augmentation). *If $(P, L \to R)$ and $V \subseteq \mathcal{V}_P$, then $(P, L \cup V \to R \cup V)$.*

Proof (soundness). Let e_1 and e_2 be embeddings of P into a relation $\mathcal{R}$ satisfying $(P, L \to R)$. If we have $e_1 =_{L \cup V} e_2$, then we have $e_1 =_L e_2$ and $e_1 =_V e_2$. By $e_1 =_L e_2$ and $(P, L \to R)$, we also have $e_1 =_R e_2$ and hence $e_1 =_{R \cup V} e_2$. □

Proposition 5 (Transitivity). *If $(P, V_1 \to V_2)$ and $(P, V_2 \to V_3)$, then $(P, V_1 \to V_3)$.*

Proof (soundness). Let e_1 and e_2 be embeddings of P into a relation $\mathcal{R}$ satisfying $(P, V_1 \to V_2)$ and $(P, V_2 \to V_3)$. If $e_1 =_{V_1} e_2$, then, by $(P, V_1 \to V_2)$, we have $e_1 =_{V_2} e_2$, and, by $(P, V_2 \to V_3)$, we have $e_1 =_{V_3} e_2$. □

Since Armstrong's axioms also hold for functional constraints, it follows that the well-known decomposition and union rules also hold for functional constraints.

Lemma 1. *Let $C_{\text{FC}} = (P, L \to R)$ be a functional constraint. Then*

$$C_{\text{FC}} \equiv \{(P, L \to \$r) \mid \$r \in R\}.$$

From now on, we assume that every functional constraint has at most one variable in its right-hand side. By Lemma 1, all our results generalize to arbitrary functional constraints.

4 Chasing Functional Constraints

For equality-generating constraints, a chase-based algorithm is known to decide implication [1]. We use the relationship between functional and equality-generating constraints described in Proposition 1 to construct a chase-based algorithm that decides implication of functional constraints, shown as Algorithm 1.

The entries in the tableau constructed in Algorithm 1 can be either constants of $\mathcal{U}$ or dedicated tableau variables, which we shall denote by capitals. These tableau variables intuitively correspond to the variables in the pattern of the target constraint. To this purpose, we assume the existence of an infinitely enumerable set $\mathfrak{V}$ of tableau variables. Further, we assume that $\mathfrak{V}$ is disjoint from both $\mathcal{U}$ and $\mathcal{V}$.[5] We generalize embeddings in a straightforward way to also allow embeddings from and to tableaux.

Algorithm 1 Chase for functional constraints

Input: A set of functional constraints $\mathcal{C} = \{(P_i, L_i \to \$r_i) \mid 1 \leq i \leq n\}$
     A functional constraint $C_{\mathrm{FC}} = (P, L \to \$r)$
Output: $\mathcal{C} \models C_{\mathrm{FC}}$
1: let $f_1, f_2 : \mathcal{V}_P \to \mathfrak{V}$ be injections with $f_1 =_L f_2$ and
          $\mathrm{range}(f_1|_{\mathcal{V}_P \setminus L}) \cap \mathrm{range}(f_2|_{\mathcal{V}_P \setminus L}) = \emptyset$
2: $\mathfrak{T} \leftarrow (f_1 \cup \mathrm{id}_{\mathcal{U}})(P) \cup (f_2 \cup \mathrm{id}_{\mathcal{U}})(P)$
3: **while** there exist functional constraint $(P_i, L_i \to \$r_i) \in \mathcal{C}$ and
          embeddings e_1, e_2 of P_i into $\mathfrak{T}$ with $e_1 =_{L_i} e_2$ and $e_1(\$r_i) \neq e_2(\$r_i)$ **do**
4:     /* *equalize $e_1(\$r_i)$ and $e_2(\$r_i)$ in $\mathfrak{T}$* */
5:     **if** $e_2(\$r_i) \in \mathfrak{V}$ **then**
6:        replace all occurrences of $e_2(\$r_i)$ in $\mathfrak{T}$ by $e_1(\$r_i)$
7:     **else if** $e_1(\$r_i) \in \mathfrak{V}$ **then**
8:        replace all occurrences of $e_1(\$r_i)$ in $\mathfrak{T}$ by $e_2(\$r_i)$
9:     **else** /* $e_1(\$r_i), e_2(\$r_i) \in \mathcal{U}$ *and* $e_1(\$r_i) \neq e_2(\$r_i)$ */
10:        **return** TRUE
11:     **end if**
12: **end while**
13: **return** $\mathfrak{T} \models C_{\mathrm{FC}}$

In Algorithm 1, we refer to lines 5–8 as *equalization steps*, to lines 9–10 as *inconsistency termination*, and to line 13 as *regular termination*. Inconsistency termination indicates that the pattern P is inconsistent with the functional constraints in $\mathcal{C}$, and, hence, that the implication under consideration is voidly true. The following example illustrates the case of inconsistency termination.

Example 6. Consider the set of functional constraints $\mathcal{C} = \{(\{(\$a, \$b)\}, \$a \to \$b)\}$. If the functional constraints in this set $\mathcal{C}$ hold on a relation $\mathcal{R}$, then no embedding of the pattern $P = \{(\$a, Constant_1), (\$a, Constant_2)\}$ with $Constant_1 \neq$

$Constant_2$ into relation $\mathcal{R}$ is possible, and, hence, every functional constraint on the pattern P holds. This is reflected by Algorithm 1: if a functional constraint on P is chased by $\mathcal{C}$, then inconsistency termination results and TRUE is returned.

Theorem 1. *Algorithm 1 is correct: it returns* TRUE *if and only if* $\mathcal{C} \models C_{\mathrm{FC}}$ *holds.*

Proof (sketch). Algorithm 1 implicitly translates the target functional constraint C_{FC} to an equality-generating constraint. Indeed, at line 2, a tableau for the pattern $P_{\mathrm{EGC}} = (f_1 \cup \mathrm{id}_{\mathcal{U}})(P) \cup (f_2 \cup \mathrm{id}_{\mathcal{U}})(P)$ is constructed. By Proposition 1, P_{EGC} is the pattern used by the equality-generating constraint equivalent to C_{FC}.

At line 3, considering two embeddings e_1 and e_2 of P_i into $\mathfrak{T}$ with $e_1 =_{L_i} e_2$ is equivalent to considering one embedding of the pattern of the equality-generating constraint equivalent to $(P_i, L_i \rightarrow \$r_i)$. Hence, Algorithm 1 can be interpreted as a chase for an equality-generating constraint with equality-generating constraints. Therefore the correctness of Algorithm 1 follows directly from the correctness of the chase algorithm for equality-generating constraints [1]. □

So, Algorithm 1 is essentially a chase algorithm for equality-generating constraints. As a consequence, it is to be expected that intermediate tableaux produced by this algorithm do not always correspond to non-trivial functional constraints. Hence, the corresponding functional constraints are not always relevant to answering $\mathcal{C} \models C_{\mathrm{FC}}$. Example 7, below, shows that this is indeed not always the case.

Example 7. We apply Algorithm 1 to the set of functional constraints

$$\mathcal{C} = \{(\{(\$a, \$b, \$c)\}, \$a \rightarrow \$c), (\{(\$a, \$b, \$c), (\$a, \$d, e)\}, \$b \rightarrow \$a)\}$$

and the target functional constraint $C_{\mathrm{FC}} = (\{(\$a, \$b, \$c), (\$a, \$b, e)\}, \$b \rightarrow \$a)$. We initially have the tableau

$$\{(A_1, B, C_1), (A_1, B, e), (A_2, B, C_2), (A_2, B, e)\}.$$

We can apply $(\{(\$a, \$b, \$c)\}, \$a \rightarrow \$c)$ to the first two tuples in this tableau, yielding the tableau

$$\{(A_1, B, e), (A_2, B, C_2), (A_2, B, e)\}.$$

We can use Proposition 1 to search for a functional constraint with such a pattern when translated to an equality-generating constraint. Let $C = (P, L \rightarrow R)$ be such a functional constraint. It is easily verified that the only way to achieve this is by relating A_1, A_2, B, C_2 to distinct variables $\$a_1, \$a_2, \$b, \$c_2 \in \mathcal{V}_P$ for which $L = \{\$a_1, \$a_2, \$b, \$c_2\}$. Since L contains all variables present in the pattern, it follows that C must be trivial. Hence, the tableau we obtained does not correspond to a functional constraint relevant to answering $\mathcal{C} \models C_{\mathrm{FC}}$.

Example 7 also illustrates the main problem of Algorithm 1. While the initial chase tableau exhibits a certain symmetry, this symmetry is lost after performing the equalization. As a consequence, only trivial functional constraints can be associated with the resulting tableau. Luckily, Algorithm 1 is non-deterministic in the equalization steps it performs. We shall take advantage of this to show the existence of a *symmetry-preserving chase*, which we define formally in Definition 7. The steps performed by symmetry-preserving chases are closely related to sound derivation steps for functional constraint in a way that shall be made precise in Section 5. Before we can introduce symmetry-preserving chases, we need some additional terminology.

Definition 5. *Let T be a tableau. A* tableau state *of T is a 4-tuple consisting of a pattern P', a set of variables $L' \subseteq \mathcal{V}_{P'}$, and injections $g_1, g_2 : \mathcal{V}_{P'} \to \mathfrak{V}$ with $g_1 =_{L'} g_2$, $\mathrm{range}(g_1|_{\mathcal{V}_{P'} \setminus L'}) \cap \mathrm{range}(g_2|_{\mathcal{V}_{P'} \setminus L'}) = \emptyset$, and $T = (g_1 \cup \mathrm{id}_{\mathcal{U}})(P') \cup (g_2 \cup \mathrm{id}_{\mathcal{U}})(P')$.*

Given a tableau T, we denote a tableau state of T such as in Definition 5 by $\mathbf{S}_T(P', L', g_1, g_2)$, this to emphasize the relationship between the tableau and the corresponding tableau state.

We can easily construct tableau states $\mathbf{S}_T(P', L', g_1, g_2)$ for every tableau T. We simply map every tableau variable from $\mathfrak{V}$ used in T to a unique variable, yielding the pattern P', and pick $L' = \mathcal{V}_{P'}$. Finally, $g_1 = g_2$ maps each variable in $\mathcal{V}_{P'}$ to the tableau variable in $\mathfrak{V}$ it represents.

Example 8. A tableau state for the tableau $\{(A_1, B, e), (A_2, B, C_2), (A_2, B, e)\}$ of Example 7 is $\mathbf{S}_{\mathfrak{T}}(P', L', g_1, g_2)$ with $P' = \{(\$a_1, \$b, e), (\$a_2, \$b, \$c_2), (\$a_2, \$b, e)\}$, $L' = \{\$a_1, \$a_2, \$b, \$c_2\}$, and $g_1 = g_2$ the injective functions mapping $\$a_1$ to A_1, $\$a_2$ to A_2, $\$b$ to B, and $\$c_2$ to C_2.

Tableau states enjoy the following useful properties.

Lemma 2. *Let $\mathbf{S}_T(P', L', g_1, g_2)$ be a state of tableau T. Then*

1. *The pattern $(g_1 \cup \mathrm{id}_{\mathcal{U}})(P')$ is isomorphic to the pattern $(g_2 \cup \mathrm{id}_{\mathcal{U}})(P')$.*
2. *For any tuple $t \in T$, also $\Phi_{g_1 \leftrightarrow g_2}(t) \in T$ and $\Phi_{g_2 \leftrightarrow g_1}(t) \in T$.*
3. *$\mathbf{S}_T(P', L', g_2, g_1)$ is also a tableau state of T.*

Proof. We have Lemma 2(1) as g_1 and g_2 are injections. Lemma 2(2) follows from Lemma 2(1), $g_1 =_{L'} g_2$, and $\mathrm{range}(g_1|_{\mathcal{V}_{P'} \setminus L'}) \cap \mathrm{range}(g_2|_{\mathcal{V}_{P'} \setminus L'}) = \emptyset$. Lemma 2(3), finally, follows immediately from Definition 5. $\square$

Observe that the initial tableau in Algorithm 1 has state $\mathbf{S}_{\mathfrak{T}}(P, L, f_1, f_2)$. We already noted that this initial tableau exhibits some symmetry due to the semantics of functional constraints. We would like that, after a sequence of equalization steps, the resulting tableau exhibits a similar symmetry. What we mean by this is made precise in Definition 6, minding that a sequence of equalization steps can be viewed as a mapping on tableau entries that maps a tableau into the tableau resulting from the equalization steps.

Definition 6. *Let m be a mapping on tableau entries, mapping a tableau T into a tableau $m(T)$. The mapping m is* symmetry-preserving *on T if there exists a tableau state $\mathbf{S}_T(P', L', g_1, g_2)$ of T and $\mathbf{S}_{m(T)}(P'', L'', g'_1, g'_2)$ of $m(T)$ such that $m((g_1 \cup \mathrm{id}_{\mathcal{U}})(P')) = (g'_1 \cup \mathrm{id}_{\mathcal{U}})(P'')$ and $m((g_2 \cup \mathrm{id}_{\mathcal{U}})(P')) = (g'_2 \cup \mathrm{id}_{\mathcal{U}})(P'')$.*

Definition 6 is visualized in Figure 2. By Lemma 2(1), $(g_1 \cup \mathrm{id}_{\mathcal{U}})(P')$ and $(g_2 \cup \mathrm{id}_{\mathcal{U}})(P')$ are isomorphic, and so are $(g'_1 \cup \mathrm{id}_{\mathcal{U}})(P'') = m((g_1 \cup \mathrm{id}_{\mathcal{U}})(P'))$ and $(g'_2 \cup \mathrm{id}_{\mathcal{U}})(P'') = m((g_2 \cup \mathrm{id}_{\mathcal{U}})(P'))$. Hence, we can say that m preserves the isomorphism between $(g_1 \cup \mathrm{id}_{\mathcal{U}})(P')$ and $(g_2 \cup \mathrm{id}_{\mathcal{U}})(P')$, explaining why we call m "symmetry-preserving".
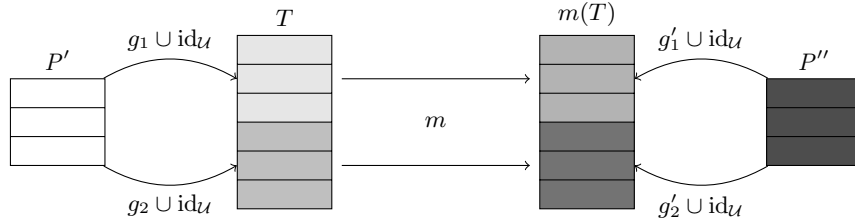

Fig. 2. Visualization of Definition 6.

Example 7 shows that not all sequences of equalization steps preserve symmetry. However, if an equalization step is possible in Algorithm 1, then also a sequence of at most two equalization steps is possible which does preserve symmetry. Moreover, all equalization steps concerned use the same constraint. This is shown next.

Theorem 2. *Let $\mathfrak{T} := T$ be the tableau of Algorithm 1 at line 3. If it is possible to perform an equalization step using the functional constraint $C_i \in \mathcal{C}$, then it is also possible to perform a sequence of at most two equalization steps, both using C_i, such that the composition of these equalization steps yields a symmetry-preserving mapping on T.*

Proof (sketch). Let $\mathbf{S}_T(P', L', g_1, g_2)$ be a tableau state of T.

If it is possible to perform an equalization with $C_i = (P_i, L_i \to \$r_i)$, then, by Lemma 2(3), we may assume, without loss of generality, that there exist terms $t_1, t_2 \in \mathcal{V}_{P'} \cup \mathcal{U}$ such that $e_1(\$r_i) = (g_1 \cup \mathrm{id}_{\mathcal{U}})(t_1)$, and either $e_2(\$r_i) = (g_1 \cup \mathrm{id}_{\mathcal{U}})(t_2)$ or $e_2(\$r_i) = (g_2 \cup \mathrm{id}_{\mathcal{U}})(t_2)$. Observe that t_1 and t_2 cannot both be constants. We now distinguish a number of cases. In each case, we suffice with providing the required sequence of at most two equalization steps and the resulting tableau $\mathfrak{T} := T'$, together with a tableau state $\mathbf{S}_{T'}(P'', L'', g'_1, g'_2)$.[6] Using the provided tableau states for T and T', it is straightforward to verify that the composition of the equalization steps is a symmetry-preserving mapping.

---

[6] From the provided tableau state $\mathbf{S}_{T'}(P'', L'', g'_1, g'_2)$, it follows implicitly what P'', L'', g'_1, and g'_2 are.

First, we consider all the cases where one of t_1 and t_2 is a variable, and the other a constant. Since the roles of e_1 and e_2 are interchangeable, we may assume, without loss of generality, that $t_1 = \$v_1$ is a variable and $t_2 = u_2$ is a constant. From the above, it follows that, in all these cases, $e_1(\$r_i) = g_1(\$v_1)$ and $e_2(\$r_i) = u_2$.

1. $\$v_1 \in L'$. Performing the equalization step using C_i, e_1, and e_2 results in the tableau $T' = \phi_{u_2 \leftarrow g_1(\$v_1)}(T)$ with state

$$\mathbf{S}_{T'}\left(\phi_{u_2 \leftarrow \$v_1}(P'), L' \setminus \{\$v_1\}, g_1|_{\mathcal{V}_{P'} \setminus \{\$v_1\}}, g_2|_{\mathcal{V}_{P'} \setminus \{\$v_1\}}\right).$$

2. $\$v_1 \notin L'$. By Lemma 2(2), the functions $\varepsilon_1 = \Phi_{g_2 \leftarrow g_1} \circ e_1$ and $\varepsilon_2 = \Phi_{g_2 \leftarrow g_1} \circ e_2$ are embeddings of P_i into T. Since $(g_1 \cup \mathrm{id}_{\mathcal{U}})(\$v_1) = g_1(\$v_1) = e_1(\$r_i) \neq e_2(\$r_i) = u_2 = (g_1 \cup \mathrm{id}_{\mathcal{U}})(u_2)$, we have, by Lemma 2(1), that $\varepsilon_1(\$r_i) = g_2(\$v_1) = (g_2 \cup \mathrm{id}_{\mathcal{U}})(\$v_1) \neq (g_2 \cup \mathrm{id}_{\mathcal{U}})(u_2) = u_2 = \varepsilon_2(\$r_i)$. The equalization step using C_i, e_1, and e_2 on T only affects tuples in $(g_1 \cup \mathrm{id}_{\mathcal{U}})(P')$ as $e_1(\$r_i) \notin \mathrm{range}(g_2)$. Hence, after the equalization step, ε_1 and ε_2 are embeddings of P_i into the resulting tableau with $\varepsilon_1(\$r_i) \neq \varepsilon_2(\$r_i)$, and, by construction, we have $\varepsilon_1 =_{L'} \varepsilon_2$. Therefore, we can perform a second equalization step using C_i, ε_1, and ε_2. Performing this second equalization step results in the tableau $T' = \phi_{u_2 \leftarrow g_1(\$v_1), u_2 \leftarrow g_2(\$v_1)}(T)$ with state

$$\mathbf{S}_{T'}\left(\phi_{u_2 \leftarrow \$v_1}(P'), L', g_1|_{\mathcal{V}_{P'} \setminus \{\$v_1\}}, g_2|_{\mathcal{V}_{P'} \setminus \{\$v_1\}}\right).$$

Next, we consider all the cases where $t_1 = \$v_1$ and $t_2 = \$v_2$ are both variables, and where $e_2(\$r_i) = g_1(\$v_2)$. Observe that $\$v_1 \neq \$v_2$ since $e_1(\$r_i) \neq e_2(\$r_i)$.

3. Both $\$v_1$ and $\$v_2$ are in L'. Performing the equalization step with C_i, e_1, and e_2 results in the tableau $T' = \phi_{g_1(\$v_1) \leftarrow g_2(\$v_2)}(T)$ with state

$$\mathbf{S}_{T'}\left(\phi_{\$v_1 \leftarrow \$v_2}(P'), L' \setminus \{\$v_2\}, g_1|_{\mathcal{V}_{P'} \setminus \{\$v_2\}}, g_2|_{\mathcal{V}_{P'} \setminus \{\$v_2\}}\right).$$

4. At least one of $\$v_1$ and $\$v_2$ is not in L'. Since the roles of e_1 and e_2 are interchangeable, we may assume, without loss of generality, that $\$v_2 \notin L'$. As in Case 2, we can perform a second equalization step following the equalization step with C_i, e_1, and e_2. Performing this second equalization step results in the tableau $T' = \phi_{g_1(\$v_1) \leftarrow g_1(\$v_2), g_2(\$v_1) \leftarrow g_2(\$v_2)}(T)$ with state

$$\mathbf{S}_{T'}\left(\phi_{\$v_1 \leftarrow \$v_2}(P'), L', g_1|_{\mathcal{V}_{P'} \setminus \{\$v_2\}}, g_2|_{\mathcal{V}_{P'} \setminus \{\$v_2\}}\right).$$

Finally, we consider all the cases where $t_1 = \$v_1$ and $t_2 = \$v_2$ are both variables, and where $e_2(\$r_i) = g_2(\$v_2)$.

5. $\$v_1 = \$v_2 = \$v$. As $g_1 =_{L'} g_2$ and $e_1(\$r_i) \neq e_2(\$r_i)$, we must have $\$v \notin L'$. The equalization step using C_i, e_1, and e_2 results in the tableau $T' = \phi_{g_1(\$v) \leftarrow g_2(\$v)}(T)$ with state

$$\mathbf{S}_{T'}\left(P', L' \cup \{\$v\}, \phi_{g_1(\$v) \leftarrow g_2(\$v)} \circ g_1, \phi_{g_1(\$v) \leftarrow g_2(\$v)} \circ g_2\right).$$

6. $v_1 \neq v_2$. By Lemma 2(2), $\varepsilon_1 = \Phi_{g_1 \leftrightarrow g_2} \circ e_1$ and $\varepsilon_2 = \Phi_{g_1 \leftrightarrow g_2} \circ e_2$ are embeddings of P_i into T. By construction and the injectivity of g_1, we have $\varepsilon_1(\$r_i) = (g_1 \cup \mathrm{id}_\mathcal{U})(\$v_1) \neq (g_1 \cup \mathrm{id}_\mathcal{U})(\$v_2) = \varepsilon_2(\$r_i)$ and $\varepsilon_1 =_{L'} \varepsilon_2$. Instead of performing the equalization using C_i, e_1, and e_2, we perform the equalization using C_i, ε_1, and ε_2. Hence, Case 6 has been reduced to Cases 3 and 4. □

We refer to each sequence of at most two equalization steps from tableau T to tableau T', considered in the proof of Theorem 2, as a *symmetry-preserving step*. We refer to the symmetry-preserving step in Case i, $1 \leq i \leq 5$, in the proof of Theorem 2 as the symmetry-preserving step of type i.[7]

Definition 7. *Executions of Algorithm 1 consisting of a sequence of symmetry-preserving steps and in which inconsistency termination occurs if no equalization steps can be performed, are called* symmetry-preserving chases.

Based on Definition 7 and on Theorem 2 we specialize Algorithm 1 to a symmetry-preserving chase algorithm, shown as Algorithm 2. Notice that we use the non-deterministic nature of Algorithm 1 to delay inconsistency termination to the latest-possible moment. By delaying inconsistency termination, we are able to perform equalization steps until no such step is possible anymore, and only then, when necessary, perform inconsistency termination.

Algorithm 2 Symmetry-preserving chase for functional constraints

---

Input: A set of functional constraints $\mathcal{C} = \{(P_i, L_i \to \$r_i) \mid 1 \leq i \leq n\}$
    A functional constraint $C_{\mathrm{FC}} = (P, L \to \$r)$
Output: $\mathcal{C} \models C_{\mathrm{FC}}$
1: let $f_1, f_2 : \mathcal{V}_P \to \mathfrak{V}$ be injections with $f_1 =_L f_2$ and
        $\mathrm{range}(f_1|_{\mathcal{V}_P \setminus L}) \cap \mathrm{range}(f_2|_{\mathcal{V}_P \setminus L}) = \emptyset$
2: $\mathfrak{T} \leftarrow (f_1 \cup \mathrm{id}_\mathcal{U})(P) \cup (f_2 \cup \mathrm{id}_\mathcal{U})(P)$
3: /* $\mathbf{S}_\mathfrak{T}(P, L, f_1, f_2)$ *is a tableau state of* $\mathfrak{T}$ */
4: **while** an equalization step can be performed using functional constraint
        $(P_i, L_i \to \$r_i) \in \mathcal{C}$ and embeddings e_1, e_2 of P_i into $\mathfrak{T}$ with
        $e_1 =_{L_i} e_2$ and $e_1(\$r_i) \neq e_2(\$r_i)$ **do**
5:    perform the corresponding symmetry-preserving step
      (cf. the proof of Theorem 2)
6: **end while**
7: **if** *inconsistency termination* **then**
8:    **return** TRUE
9: **else**
10:    **return** $\mathfrak{T} \models C_{\mathrm{FC}}$
11: **end if**

Theorem 2 now immediately yields the following.

---

[7] We have no symmetry-preserving step of type 6, as Case 6 in the proof of Theorem 2 has been reduced to Cases 3 and 4.

Corollary 1. *Algorithm 2 is correct: it returns* TRUE *if and only if* $\mathcal{C} \models C_{\mathrm{FC}}$
holds.

5 Axiomatization for the Functional Constraints

Let $\mathcal{C}$ be a set of functional constraints and let $C_{\mathrm{FC}} = (P, L \to \$r)$ be a single
functional constraint for which $\mathcal{C} \models C_{\mathrm{FC}}$. By simulating a symmetry-preserving
chase for $\mathcal{C} \models C_{\mathrm{FC}}$ (Algorithm 2), by a derivation of functional constraints us-
ing sound derivation rules, we construct an axiomatization for the functional
constraints which must be complete by Corollary 1.

First, we consider the (base) cases where the chase terminates immediately
without performing symmetry-preserving steps. By the *restricted reflexivity ax-
iom*, below, we mean the specialization of the reflexivity axiom in which only
functional constraints are derived with at most one variable in the right-hand
side.

Lemma 3. *If only regular termination is possible in a symmetry-preserving
chase for* $\mathcal{C} \models C_{\mathrm{FC}}$, *then* C_{FC} *can be derived using the restricted reflexivity axiom.*

Proof. Consider a symmetry-preserving chase for $\mathcal{C} \models C_{\mathrm{FC}}$. If initially only
regular termination is possible, then this chase is also a successful symmetry-
preserving chase for $\emptyset \models C_{\mathrm{FC}}$. It follows that $\mathfrak{T} \models C_{\mathrm{FC}}$, with $\mathfrak{T}$ the initial tableau
constructed in lines 1–2 of Algorithm 2. This implies $f_1(\$r) = f_2(\$r)$, which in
turn implies $\$r \in L$. Hence, C_{FC} can be derived using the restricted reflexivity
axiom. □

For the case where initially only inconsistency termination is possible, we
introduce the inconsistency axiom, of which we prove the soundness next.

Proposition 6 (Inconsistency). *If* $(P', L' \to \$r')$, *if there exist two embed-
dings of* P' *into a pattern* P *which agree on* $L' \in \mathcal{V}_{P'}$ *and map* $\$r'$ *to different
constants of* $\mathcal{U}$, *and if* $\$r \in \mathcal{V}_P$, *then* $(P, L \to \$r)$.

Proof (soundness). Let e be an embedding of P into a relation $\mathcal{R}$ satisfying
$(P', L' \to \$r')$. Let h_1 and h_2 be two embeddings of P' into P satisfying the
conditions of Proposition 6. Then, clearly, $\varepsilon_1 = e \circ h_1$ and $\varepsilon_2 = e \circ h_2$ are
embeddings of P' into $\mathcal{R}$ with $\varepsilon_1 =_{L'} \varepsilon_2$ and $\varepsilon_1(\$r_i) \neq \varepsilon_2(\$r_i)$. Hence, if there is
an embedding of P into $\mathcal{R}$, then there exist two embeddings e_1 and e_2 of P' into
$\mathcal{R}$ that agree on L', but not on $\$r'$. Hence, embeddings e_1 and e_2 show that $\mathcal{R}$
violates the functional constraint $(P', L' \to \$r')$, a contradiction. We conclude
that there is no embedding of P into $\mathcal{R}$, as a consequence of which $\mathcal{R}$ voidly
satisfies $(P, L \to \$r)$. □

We observe that the inconsistency axiom can be used in Example 6 to derive
$(P, L \to \$r)$ from $\mathcal{C}$. We now generalize this observation.

Lemma 4. *If initially only inconsistency termination is possible in a symmetry-
preserving chase for* $\mathcal{C} \models C_{\mathrm{FC}}$, *then* C_{FC} *can be derived from* $\mathcal{C}$ *using the incon-
sistency axiom.*

Proof. Consider a symmetry-preserving chase for $\mathcal{C} \models C_{\text{FC}}$. If inconsistency termination is possible, then there exists a functional constraint $C_i = (P_i, L_i \to \$r_i) \in \mathcal{C}$ and embeddings e_1, e_2 of P_i into $\mathfrak{T}$ with $e_1 =_{L_i} e_2$, $e_1(\$r_i) \neq e_2(\$r_i)$, and $e_1(\$r_i), e_2(\$r_i) \in \mathcal{U}$. The embeddings e_1 and e_2 map P_i into $\mathfrak{T}$ and the function $(f_1^{-1} \cup f_2^{-1} \cup \text{id}_\mathcal{U})$, which is well defined, maps $\mathfrak{T}$ into P. Hence, $h_1 = (f_1^{-1} \cup f_2^{-1} \cup \text{id}_\mathcal{U}) \circ e_1$ and $h_2 = (f_1^{-1} \cup f_2^{-1} \cup \text{id}_\mathcal{U}) \circ e_2$ are embeddings of P_i into P with $h_1 =_{L_i} h_2$, $h_1(\$r_i) \neq h_2(\$r_i)$, and $h_1(\$r_i), h_2(\$r_i) \in \mathcal{U}$. Hence, C_{FC} can be derived from C_i using the inconsistency axiom. $\square$

Next, consider the case where the chase for $\mathcal{C} \models C_{\text{FC}}$ initially performs a symmetry-preserving step. We introduce the axioms *pattern-modification* and *left-modification* to deal with this case.

Proposition 7 (Pattern-modification). *Let P be a pattern, $L \subseteq \mathcal{V}_P$, $t \in \mathcal{V}_P \cup \mathcal{U}$, and $\$r, \$v \in \mathcal{V}_P$. If $(P', L' \to \$r')$, and $(\phi_{t \leftrightarrow \$v}(P), \phi_{t \leftrightarrow \$v}(L) \cap \mathcal{V}_P \to \{\phi_{t \leftrightarrow \$v}(\$r)\} \cap \mathcal{V}_P)$, and if there exists two embeddings of P' into P which agree on L' and map $\$r'$ to t and $\$v$, respectively, then $(P, L \to \$r)$.*

Proof (soundness). Let e be an embedding of P into a relation $\mathcal{R}$ satisfying $(P', L' \to \$r')$ and $(\phi_{t \leftrightarrow \$v}(P), \phi_{t \leftrightarrow \$v}(L) \cap \mathcal{V}_P \to \{\phi_{t \leftrightarrow \$v}(\$r)\} \cap \mathcal{V}_P)$. Let h_1 and h_2 be two embeddings of P' into P satisfying the conditions of Proposition 7. Then, $\varepsilon_1 = e \circ h_1$ and $\varepsilon_2 = e \circ h_2$ are embeddings of P' into $\mathcal{R}$ with $\varepsilon_1 =_{L'} \varepsilon_2$. By $(P', L' \to \$r')$, we have $\varepsilon_1(\$r') = \varepsilon_2(\$r')$, and hence $e(t) = e(\$v)$. Hence, $e|_{\text{domain}(e) \backslash \{\$v\}}$ is an embedding of $\phi_{t \leftrightarrow \$v}(P)$ into $\mathcal{R}$.

Now, let e_1 and e_2 be two embeddings of P into $\mathcal{R}$ with $e_1 =_L e_2$. From the above, $\varepsilon_1 = e_1|_{\text{domain}(e_1) \backslash \{\$v\}}$ and $\varepsilon_2 = e_2|_{\text{domain}(e_2) \backslash \{\$v\}}$ are embeddings of $\phi_{t \leftrightarrow \$v}(P)$ into $\mathcal{R}$ satisfying $\varepsilon_1 =_{\phi_{t \leftrightarrow \$v}(L)} \varepsilon_2$, and hence, also $\varepsilon_1 =_{\phi_{t \leftrightarrow \$v}(L) \cap \mathcal{V}_P} \varepsilon_2$. By $(\phi_{t \leftrightarrow \$v}(P), \phi_{t \leftrightarrow \$v}(L) \cap \mathcal{V}_P \to \{\phi_{t \leftrightarrow \$v}(\$r)\} \cap \mathcal{V}_P)$, we have $\varepsilon_1 =_{\{\phi_{t \leftrightarrow \$v}(\$r)\} \cap \mathcal{V}_P} \varepsilon_2$, and, hence, we have $\varepsilon_1 =_{\{\phi_{t \leftrightarrow \$v}(\$r)\}} \varepsilon_2$. As $e_1(t) = e_1(\$v)$ and $e_2(t) = e_2(\$v)$, we also have $e_1(\$r) = e_2(\$r)$, even if $\$r = \$v$. $\square$

Generally speaking, the pattern-modification axiom modifies the pattern of a functional constraint. More specifically, the axiom generalizes the pattern of a constraint due to constraints imposed by other functional constraints.

Example 9. Consider the set of functional constraints

$$\mathcal{C} = \{(\{(\$a, \$b, \$c)\}, \$a \to \$c), (\{(\$a, \$b, \$c), (\$a, \$b, e)\}, \$b \to \$a)\}$$

and the target functional constraint $C_{\text{FC}} = (\{(\$a, \$b, e)\}, \$b \to \$a)$. We can derive C_{FC} from $\mathcal{C}$ by using the the embeddings h_1 and h_2 mapping $\{(\$a, \$b, \$c)\}$ to $\{(\$a, \$b, \$c)\}$ and $\{(\$a, \$b, e)\}$, respectively, and by picking $t = e$ and $\$v = \$c$. Indeed, due to the constraint imposed by $(\{(\$a, \$b, \$c)\}, \$a \to \$c)$, we are able to generalize $(\{(\$a, \$b, \$c), (\$a, \$b, e)\}, \$b \to \$a)$ to C_{FC}.

Proposition 8 (Left-modification). *Let P be a pattern, $L \subseteq \mathcal{V}_P$, $\$v \in \mathcal{V}_P$, and let $i_1, i_2 : \mathcal{V}_P \to \mathcal{V}$ be injective functions with $i_1(\$v) \neq i_2(\$v)$, $i_1 =_L i_2$, and $\text{range}(i_1|_{\mathcal{V}_P \backslash L}) \cap \text{range}(i_2|_{\mathcal{V}_P \backslash L}) = \emptyset$. If $(P', L' \to \$r')$, and $(P, L \cup \{\$v\} \to \$r)$, and if there exist two embeddings from P' into $(i_1 \cup \text{id}_\mathcal{U})(P) \cup (i_2 \cup \text{id}_\mathcal{U})(P)$ which agree on L' and map $\$r'$ to $i_1(\$v)$ and $i_2(\$v)$, respectively, then $(P, L \to \$r)$.*

Proof (soundness). Let e_1 and e_2 be two embeddings of P into a relation $\mathcal{R}$ satisfying $(P', L' \to \$r')$, $(P, L \cup \{\$v\} \to \$r)$, and $e_1 =_L e_2$. Let h_1 and h_2 be two embeddings of P' into $(i_1 \cup \mathrm{id}_\mathcal{U})(P) \cup (i_2 \cup \mathrm{id}_\mathcal{U})$ satisfying the conditions of Proposition 8. Since $i_1 =_L i_2$, $e_1 =_L e_2$, and $i_1 \cup \mathrm{id}_\mathcal{U}$ and $i_2 \cup \mathrm{id}_\mathcal{U}$ are injections whose range only overlap on $L \cup \mathcal{U}$, the function $f = \Phi_{e_1 \hookleftarrow i_1 \cup \mathrm{id}_\mathcal{U}} \circ \Phi_{e_2 \hookleftarrow i_2 \cup \mathrm{id}_\mathcal{U}}$ is well-defined. Hence, the functions $\varepsilon_1 = f \circ h_1$ and $\varepsilon_2 = f \circ h_2$ are embeddings of P' into $\mathcal{R}$ with $\varepsilon_1 =_{L'} \varepsilon_2$. By construction, we have $\varepsilon_1(\$r') = e_1(\$v)$ and $\varepsilon_2(\$r') = e_2(\$v)$. Hence, by $(P', L' \to \$r')$, we have $e_1(\$v) = e_2(\$v)$, and thus $e_1 =_{L \cup \{\$v\}} e_2$. By $(P, L \cup \{\$v\} \to \$r)$ and $e_1 =_{L \cup \{\$v\}} e_2$, we conclude $e_1(\$r) = e_2(\$r)$. $\qquad\square$

The left-modification axiom generalizes a functional constraint by removing a variable from its left-hand side. This as a consequence of constraints imposed by other functional constraints.

Example 10. Consider the set of functional constraints

$$\mathcal{C} = \{(\{(\$a, \$b, \$c), (d, \$e, \$f)\}, \$c \to \$f),$$
$$(\{(\$a, \$b, \$c), (d, \$e, \$f)\}, \{\$a, \$f\} \to \$b)\}$$

and the target functional constraint $C_{\mathrm{FC}} = (\{(\$a, \$b, \$c), (d, \$e, \$f)\}, \$a \to \$b)$. We pick i_1 and i_2 such that:

$$(i_1 \cup \mathrm{id}_\mathcal{U})(\{(\$a, \$b, \$c), (d, \$e, \$f)\}) = \{(\$a, \$b_1, \$c_1), (d, \$e_1, \$f_1)\}$$
$$(i_2 \cup \mathrm{id}_\mathcal{U})(\{(\$a, \$b, \$c), (d, \$e, \$f)\}) = \{(\$a, \$b_2, \$c_2), (d, \$e_2, \$f_2)\}.$$

We can derive C_{FC} from $\mathcal{C}$ by picking the embeddings h_1 and h_2 such that:

$$h_1(\{(\$a, \$b, \$c), (d, \$e, \$f)\}) = \{(\$a, \$b_1, \$c_1), (d, \$e_1, \$f_1)\}$$
$$h_2(\{(\$a, \$b, \$c), (d, \$e, \$f)\}) = \{(\$a, \$b_1, \$c_1), (d, \$e_2, \$f_2)\}.$$

Indeed, due to the constraint imposed by $(\{(\$a, \$b, \$c), (d, \$e, \$f)\}, \$c \to \$f)$, we are able to generalize $(\{(\$a, \$b, \$c), (d, \$e, \$f)\}, \{\$a, \$f\} \to \$b)$ to C_{FC}. We notice that there is a relation between the left-modification axiom and the well-known multivalued dependencies [22]. In this example, the possible embeddings of the pattern $\{(\$a, \$b, \$c), (d, \$e, \$f)\}$ can be represented by a relational table T with schema $R(A, B, C, E, F)$. Due to $\mathcal{C}$, the functional dependencies $C \to F$ and $AF \to B$ hold on T. Due to the construction of T, also the multivalued dependency $A \twoheadrightarrow EF$ holds. Indeed, by using well-known derivation rules for functional dependencies and multivalued dependencies, we conclude $A \to B$.

We claim that the pattern-modification axiom simulates the symmetry-preserving steps of type 1–4, and the left-modification axiom the symmetry-preserving steps of type 5. Before proving that this is indeed the case, we introduce an auxiliary derivation rule. We emphasize that this rule is not part of our axiomatization. We shall only use its soundness to simplify the proof of Lemma 6.

Lemma 5 (Embedding). *If $(P', L' \to \$r')$ and h is an embedding from P' into P, then $(P, h(L') \cap \mathcal{V}_P \to \{h(\$r')\} \cap \mathcal{V}_P)$.*

Proof (soundness). Let e_1 and e_2 be embeddings of P into a relation $\mathcal{R}$ satisfying $(P', L' \to \$r')$. Then, $\varepsilon_1 = e_1 \circ h$ and $\varepsilon_2 = e_2 \circ h$ are embeddings of P' into $\mathcal{R}$. If $e_1 =_{h(L') \cap \mathcal{V}_P} e_2$, then $e_1 =_{h(L')} e_2$ and $\varepsilon_1 =_{L'} \varepsilon_2$, as embeddings always agree on constants. By $(P', L' \to \$r')$, we have $\varepsilon_1(\$r') = \varepsilon(\$r')$. As a consequence, we have $e_1 =_{h(\{\$r'\})} e_2$ and, hence, also $e_1 =_{h(\{\$r'\}) \cap \mathcal{V}_P} e_2$. $\qquad\square$

The embedding rule explicitly maps functional constraints to different patterns, whereas the chase algorithm implicitly uses embeddings to deal with different patterns.

Example 11. If $(\{(\$a, \$b)\}, \$a \to \$b)$ holds, then trivially also $(\{(\$c, \$d)\}, \$c \to \$d)$ holds. We can derive $(\{(\$c, \$d)\}, \$c \to \$d)$ from $(\{(\$a, \$b)\}, \$a \to \$b)$ by using the embedding rule with the embedding that maps $\$a$ to $\$c$ and $\$b$ to $\$d$.

We now prove that symmetry-preserving steps can indeed be simulated by the pattern-modification and left-modification axioms.

Lemma 6. *Consider a successful symmetry-preserving chase for $\mathcal{C} \models C_{\mathrm{FC}}$. If the chase starts with a symmetry-preserving step, using the functional constraint $C_i \in \mathcal{C}$ and resulting in tableau T' with tableau state $\mathbf{S}_{T'}(P', L', g_1, g_2)$, then there exists a functional constraint $C = (P', L' \to \$r')$ such that*

1. *the remainder of the chase starting from tableau T' is a successful symmetry-preserving chase for $\mathcal{C} \models C$.*
2. *we can derive C_{FC} from C_i and C using the pattern-modification and left-modification axioms.*

Proof. Let $\mathfrak{T} := T$ be the initial tableau in Algorithm 2. We assume that the initial symmetry-preserving step using $C_i = (P_i, L_i \to \$r_i)$ equalizes with the embeddings e_1 and e_2 satisfying $e_1 =_{L_i} e_2$ and $e_1(\$r_i) \neq e_2(\$r_i)$. The embeddings e_1 and e_2 map P_i into T and the function $(f_1^{-1} \cup f_2^{-1} \cup \mathrm{id}_{\mathcal{U}})$, which is well defined, maps T into P. Hence, $h_1 = (f_1^{-1} \cup f_2^{-1} \cup \mathrm{id}_{\mathcal{U}}) \circ e_1$ and $h_2 = (f_1^{-1} \cup f_2^{-1} \cup \mathrm{id}_{\mathcal{U}}) \circ e_2$ are embeddings of P_i into P with $h_1 =_{L_i} h_2$. Since the roles of f_1 and f_2 are interchangeable, we may assume, without loss of generality, that $e_1(\$r_1) = (f_1 \cup \mathrm{id}_{\mathcal{U}})(t_1)$ and either $e_2(\$r_1) = (f_1 \cup \mathrm{id}_{\mathcal{U}})(t_2)$ or $e_2(\$r_1) = (f_2 \cup \mathrm{id}_{\mathcal{U}})(t_2)$. Here, t_1 and t_2 are terms of $\mathcal{V}_P \cup \mathcal{U}$ which are not both constants. We now distinguish two cases.

1. *The symmetry-preserving step is of type 1–4.* Without loss of generality, we may assume that $t_2 = \$v_2 \in \mathcal{V}_P$. The symmetry-preserving step results in a tableau $\mathfrak{T} := T' = \phi_{(f_1 \cup \mathrm{id}_{\mathcal{U}})(t_1) \leftarrow f_1(\$v_2), (f_2 \cup \mathrm{id}_{\mathcal{U}})(t_1) \leftarrow f_2(\$v_2)}(T)$ with state

   $$\mathbf{S}_{T'}\left(\phi_{t_1 \leftarrow \$v_2}(P), L \setminus \{\$v_2\}, f_1|_{\mathcal{V}_P \setminus \{\$v_2\}}, f_2|_{\mathcal{V}_P \setminus \{\$v_2\}}\right).$$

   Let $C = (\phi_{t_1 \leftrightarrow \$v_2}(P), \phi_{t_1 \leftrightarrow \$v_2}(L) \cap \mathcal{V}_P \to \{\phi_{t_1 \leftrightarrow \$v_2}(\$r)\} \cap \mathcal{V}_P)$. Clearly, T' is an initial tableau for the symmetry-preserving chase for $\mathcal{C} \models C$. It

follows that the remainder of the chase for $\mathcal{C} \models C_{\mathrm{FC}}$ is a successful chase for $\mathcal{C} \models C$, as C can be derived from C_{FC} using the embedding rule with embedding $\phi_{t \leftrightarrow \$w}$. By construction of h_1 and h_2, we have $h_1(\$r_i) = t_1$ and $h_2(\$r_i) = \$v_2$. Hence, C_{FC} can be derived from C_i and C using the pattern-modification axiom.

2. *The symmetry-preserving step is of type 5.* Then $t_1 = t_2 = \$v \in \mathcal{V}_P$. The symmetry-preserving step results in a tableau $\mathfrak{T} = T' = \phi_{f_1(\$v) \leftrightarrow f_2(\$v)}(T)$ with state

$$\mathbf{S}_{T'}\left(P, L \cup \{\$v\}, \phi_{f_1(\$v) \leftrightarrow f_2(\$v)} \circ f_1, \phi_{f_1(\$v) \leftrightarrow f_2(\$v)} \circ f_2\right).$$

Let $C = (P, L \cup \{\$v\} \to \$r)$. Clearly, T' is an initial tableau for the symmetry-preserving chase for $\mathcal{C} \models C$. It follows that the remainder of the chase for $\mathcal{C} \models C_{\mathrm{FC}}$ is a successful chase for $\mathcal{C} \models C$ as C can be derived from C_{FC} using a straightforward application of the reflexivity and transitivity axioms. Observe that $t_1 = f_1(\$v)$ and $t_2 = f_2(\$v)$ together with the embeddings e_1 and e_2 satisfy the conditions of Proposition 8, which allows the derivation of C_{FC} from C_i and C using the left-modification axiom. $\square$

As a consequence of Corollary 1, Lemmas 3–6 yield a sound and complete axiomatization of the functional constraints.

Theorem 3. *The restricted reflexivity, inconsistency, pattern-modification, and left-modification axioms constitute an axiomatization for the functional constraints with at most one variable in their right-hand side.*

Proof. We have already proven soundness of the axioms and it is straightforward that the axioms are recursive, hence we only need to verify that the axioms are complete. Let $\mathcal{C}$ be a set of functional constraints and C_{FC} be a functional constraint with $\mathcal{C} \models C_{\mathrm{FC}}$. By Corollary 1, there exists a successful symmetry-preserving chase for $\mathcal{C} \models C_{\mathrm{FC}}$. We must prove, which we shall do by induction on the number of symmetry-preserving steps performed in this chase, that $\mathcal{C} \vdash C_{\mathrm{FC}}$. The base case is that no symmetry-preserving steps are performed, i.e., that the chase terminates immediately. Then $\mathcal{C} \vdash C_{\mathrm{FC}}$ follows from Lemma 3 and 4.

As inductive hypothesis, we assume that the existence of a successful symmetry-preserving chase for $\mathcal{C}' \models C'_{\mathrm{FC}}$ with $i \geq 0$ symmetry-preserving steps ($\mathcal{C}'$ a set of functional constraints and C'_{FC} a single functional constraint) yields $\mathcal{C}' \vdash C'_{\mathrm{FC}}$. For the inductive step, assume that the successful symmetry-preserving chase for $\mathcal{C} \models C_{\mathrm{FC}}$ has $i + 1$ symmetry-preserving steps. Assume that the first symmetry-preserving step uses $C_i \in \mathcal{C}$. By Lemma 6, there exists a functional constraint $C = (P', L' \to \$r')$ such that $\{C_i, C\} \vdash C_{\mathrm{FC}}$ and such that the remainder of the chase is a successful symmetry-preserving chase for $\mathcal{C} \models C$. As this chase has only i symmetry-preserving steps, the inductive hypothesis yields $\mathcal{C} \vdash C$. We thus conclude that $\mathcal{C} \vdash C_{\mathrm{FC}}$, which completes the proof. $\square$

Using Lemma 1 we generalize Theorem 3 to functional constraints with arbitrary sets of variables in their right-hand side.

Corollary 2. *The inconsistency, pattern-modification, and left-modification axioms together with the reflexivity, augmentation, and transitivity axioms constitute an axiomatization for the functional constraints.*

Moreover, we have the following (proof omitted).

Theorem 4. *The axiomatization of the functional constraints is no longer complete if one of the axioms reflexivity, augmentation, transitivity, inconsistency, pattern-modification, or left-modification is removed.*

6 Conclusions and directions for future work

Starting from functional and equality-generating constraints for the RDF data model, we studied functional constraints on arbitrary relations. As our first result, we proved the existence of a symmetry-preserving chase for the functional constraints. Using the symmetry-preserving chase, we derived a sound and complete axiomatization for the functional constraints. This solves a major open problem in the work on functional constraints for the RDF data model.

We believe that our work provides a promising formal basis for reasoning about functional constraints. As for future work, one remaining open problem is the existence of Armstrong relations [15, 19] for the functional constraints. Another avenue of research concerns generalizations of functional constraints. In particular, adding constants to the right-hand side of functional constraints would result in a very powerful class of constraints that generalizes both the functional constraints and the conditional functional dependencies [17]. Finally, it is unknown what the complexity of working with functional constraints is, as compared with the functional dependencies and equality-generating constraints.

References

1. Akhtar, W., Cortés-Calabuig, Á., Paredaens, J.: Constraints in RDF. In: Semantics in Data and Knowledge Bases. Volume 6834 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 23–39
2. Codd, E.F.: Relational completeness of data base sublanguages. Technical Report RJ 987, IBM Research Laboratory, San Jose, California (1972)
3. Codd, E.F.: Recent investigations in relational data base systems. In: Information Processing 74. (1974) 1017–1021
4. Beeri, C., Vardi, M.: The implication problem for data dependencies. In: Automata, Languages and Programming. Volume 115 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1981) 73–85
5. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
6. Lausen, G., Meier, M., Schmidt, M.: SPARQLing constraints for RDF. In: Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology. EDBT '08 (2008) 499–509

7. Hartmann, S., Link, S.: More functional dependencies for XML. In: Advances in Databases and Information Systems. Volume 2798 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2003) 355–369

8. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.C.: Keys for XML. Computer Networks **39**(5) (2002) 473–487

9. Hartmann, S., Link, S.: Efficient reasoning about a robust XML key fragment. ACM Transactions on Database Systems **34**(2) (2009) 10:1–10:33

10. Vincent, M.W., Liu, J., Mohania, M.: The implication problem for 'closest node' functional dependencies in complete XML documents. Journal of Computer and System Sciences **78**(4) (2012) 1045–1098

11. Arenas, M., Libkin, L.: A normal form for XML documents. ACM Transactions on Database Systems **29**(1) (2004) 195–232

12. Calbimonte, J.P., Porto, F., Keet, C.M.: Functional dependencies in OWL ABOX. In: XXIV Simpósio Brasileiro de Banco de Dados. (2009) 16–30

13. Yu, Y., Heflin, J.: Extending functional dependency to detect abnormal data in RDF graphs. In: The Semantic Web ISWC 2011. Volume 7031 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 794–809

14. Cortés-Calabuig, A., Paredaens, J.: Semantics of constraints in RDFS. In: Proceedings of the 6th Alberto Mendelzon International Workshop on Foundations of Data Management. (2012) 75–90

15. Fagin, R.: Horn clauses and database dependencies. Journal of the ACM **29**(4) (1982) 952–985

16. Wijsen, J.: Database repairing using updates. ACM Transactions on Database Systems **30**(3) (2005) 722–768

17. Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for capturing data inconsistencies. ACM Transactions on Database Systems **33**(2) (2008) 6:1–6:48

18. He, Q., Ling, T.W.: Extending and inferring functional dependencies in schema transformation. In: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management. CIKM '04, ACM (2004) 12–21

19. Armstrong, W.W.: Dependency structures of data base relationships. In: Information Processing 74. (1974) 580–583

20. Aho, A.V., Beeri, C., Ullman, J.D.: The theory of joins in relational databases. ACM Transactions on Database Systems **4**(3) (1979) 297–314

21. Beeri, C., Vardi, M.Y.: A proof procedure for data dependencies. Journal of the ACM **31**(4) (1984) 718–741

22. Beeri, C., Fagin, R., Howard, J.H.: A complete axiomatization for functional and multivalued dependencies in database relations. In: Proceedings of the 1977 ACM SIGMOD International Conference on Management of Data. SIGMOD '77 (1977) 47–61