Implication and axiomatization of functional and constant constraints

Annals of Mathematics and Artificial Intelligence manuscript No.
(will be inserted by the editor)

Implication and Axiomatization of Functional and Constant Constraints

Jelle Hellings · Marc Gyssens · Jan Paredaens · Yuqing Wu

Received: date / Accepted: date

Abstract Akhtar et al. introduced equality-generating constraints and functional constraints as a first step towards dependency-like integrity constraints for RDF data [3]. Here, we focus on functional constraints. Since the usefulness of functional constraints is not limited to the RDF data model, we study the functional constraints in the more general setting of relations with arbitrary arity. We further introduce constant constraints and study the functional and constant constraints combined.

Our main results are sound and complete axiomatizations for the functional and constant constraints, both separately and combined. These axiomatizations are derived using the chase algorithm for equality-generating constraints. For derivations of constant constraints, we show how every chase step can be simulated by a bounded number of applications of inference rules. For derivations of functional constraints, we show that the chase algorithm can be normalized to a more specialized symmetry-preserving chase algorithm performing so-called symmetry-preserving steps. We then show how each symmetry-preserving step can

This is a revised and extended version of the paper 'Implication and Axiomatization of Functional Constraints on Patterns with an Application to the RDF Data Model' presented at the 8th International Symposium on Foundations of Information and Knowledge Systems, Bordeaux, France (FOIKS 2014) [25].

Yuqing Wu carried out part of her work during a sabbatical visit to Hasselt University with a Senior Visiting Postdoctoral Fellowship of the Research Foundation Flanders (FWO).

Jelle Hellings · Marc Gyssens
Hasselt University and Transnational University of Limburg, Faculty of Sciences
Martelarenlaan 42, 3500 Hasselt, Belgium
E-mail: {jelle.hellings, marc.gyssens}@uhasselt.be

Jan Paredaens
University of Antwerp, Department of Mathematics and Computer Science
Campus Middelheim, Bldg. G, Middelheimlaan 1, 2020 Antwerp, Belgium
E-mail: jan.paredaens@uantwerpen.be

Yuqing Wu
Pomona College, Computer Science Department
185 E. 6th St. Claremont, CA 91711
E-mail: melanie.wu@pomona.edu

be simulated by a bounded number of applications of inference rules. The axiomatization for functional constraints is in particular applicable to the RDF data model, solving a major open problem of Akhtar et al.

Keywords Functional constraints · Constant 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 us to reason over the data and identify inconsistencies. As such, formal constraints play a major role in database management systems that automatically maintain integrity of the data and optimize query evaluation. In this work, we study what we call constant-functional constraints. These constant-functional constraints are a generalization of the well-known functional dependencies of Codd [13], the conditional functional dependencies of Fan et al. [21], and the functional constraints of Akhtar et al. [3,15,25]. More concretely, the constant-functional constraints can be characterized as the union of the functional constraints applied to arbitrary relations, as presented by Hellings et al. [25], and the constant constraints, which we introduce here.

Functional constraints have the form

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

where P specifies a pattern in the 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 data match the pattern and are equal on L, then they must also be equal on R. Example 1 illustrates this for ternary RDF data.

Example 1 Consider the family tree shown in Figure 1.

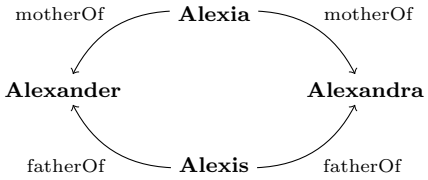

Fig. 1 Simplified visualization of an RDF representation of a small family tree.

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

Functional constraints allow the expression of not only the traditional functional dependencies, but also of context-dependent functional dependencies that only apply to a part of the data. In the context of Example 1, a context-dependent

functional dependency could be the constraint *child* $\rightarrow$ *parent* restricted to *motherOf* triplets.

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 2 Let $Edge(from, to)$ be a binary relation schema representing the edge relation 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 at 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.

With functional constraints, one cannot specify that data entries matching a variable in some pattern should have a specified constant value. To extend the scope of the constraints under consideration, we introduce constant constraints and study functional and constant constraints combined and refer to them as constant-functional constraints. Constant constraints have the form

$$(P, E),$$

where P is a pattern and E is a finite set of constant equalities of the form $(c = \$v)$, with c a constant and $\$v$ a variable occurring in the pattern P.

Example 3 Let $PI(name, country, cc, phone)$ be the schema of a relation storing personal information, in which the attribute *cc* provides the *country code* that should be used to call the phone number *phone*. The constant constraint $(\{(\$n, BE, \$c, \$p)\}, \{(\text{`32'} = \$c)\})$ specifies that people from Belgium have a Belgian phone number.

For the functional dependencies in the relational data model, a sound and complete axiomatization is already long known [5], and, more recently, Akhtar et al. presented a sound and complete axiomatization for the equality-generating constraints in the RDF data model [3]. Since the constant-functional constraints are subsumed by equality-generating constraints, this axiomatization can also be used for the inference of constant-functional constraints only. In this case, intermediate inference steps can generate equality-generating constraints that are not necessarily equivalent to constant-functional constraints, unfortunately. For the functional constraints, Akhtar et al. identified the existence of a sound and complete axiomatization (not involving other types of constraints) as a major open problem. On the one hand, the Armstrong axiomatization for the functional dependencies [5] 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 constant constraints, the functional constraints, and the constant-functional constraints

over relations of arbitrary arity. The constraints used by all intermediate inference steps allowed by these axiomatizations are constant constraints, functional constraints, and constant-functional constraints, respectively. 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. [3].

Our axiomatization is derived using the chase algorithm for equality-generating constraints [3], which is a variation of the standard chase algorithm [2, 8]. For derivations of constant constraints, we show how every chase step can be simulated by a bounded number of applications of inference rules. For derivations of functional constraints, such a direct simulation of chase steps is not straightforward. The key insight we use to circumvent this problem is that the chase algorithm, when applied to decide if a functional constraint holds, can be normalized to a more specialized, symmetry-preserving, chase algorithm. 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, by using so-called symmetry-preserving steps. We then show how each symmetry-preserving step can be simulated by a bounded number of applications of inference rules. The axiomatization for the constant-functional constraints follows from these simulations. Axiomatizations for only the constant constraints and only the functional constraints can be derived in a similar manner.

This is a revised and extended version of Hellings et al. [25]. Compared to Hellings et al., we present a slightly simplified sound and complete axiomatization of the functional constraints. Additionally, we present the constant constraints and provide sound and complete axiomatizations of constant constraints and of constant-functional constraints.

Organization. In Section 2, we present the necessary definitions used throughout this paper. In Section 3, we discuss the constant-functional constraints and equality-generating constraints. In Section 4, we present the chase algorithm for equality-generating constraints and specialize it to constant-functional constraints. In Section 5, we propose an axiomatization for the constant-functional constraints. In Section 6, we look at the relationships between the constant-functional constraints and previously introduced classes of dependencies. In Section 7, we summarize our findings and discuss directions for future work.

2 Preliminaries

Functional and equality-generating constraints [3] 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 $\mathcal{U}_P$, $\mathcal{V}_P$, and $\mathcal{T}_P$ denote the set of all constants, variables, and terms in P, respectively. A pattern P with $\mathcal{V}_P = \emptyset$ is usually referred to as a *relation*.

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 *term-based renaming function* $\phi_{a_1 \leftrightarrow b_1, \ldots, a_i \leftrightarrow b_i}$, $a_1, b_1, \ldots, a_i, b_i \in \mathcal{T}$, is the function on $\mathcal{T}$ for which $\phi_{a_1 \leftrightarrow b_1, \ldots, a_i \leftrightarrow b_i}(b_j) = a_j$, $j = 1, \ldots, i$, and which is the identity elsewhere. Likewise, the *function-based renaming function* $\Phi_{f \leftrightarrow g}$, with f a function and g an injective function with $\mathrm{domain}(f) = \mathrm{domain}(g)$, is the function on $\mathcal{T}$ for which $\Phi_{f \leftrightarrow g}(g(t)) = f(t), t \in \mathrm{domain}(f)$, 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 $e(P) \subseteq Q$.

For constraints for which this is relevant, we denote "pattern P *satisfies* constraint C" by $P \models C$. Likewise, for a set of constraints $\mathcal{S}$, we write $P \models \mathcal{S}$ to denote that $P \models C$ for all $C \in \mathcal{S}$.[1]

We say that "set of constraints $\mathcal{S}$ *implies* set of constraints $\mathcal{S}'$", denoted by $\mathcal{S} \models \mathcal{S}'$, if, for every relation $\mathcal{R}$, $\mathcal{R} \models \mathcal{S}$ implies $\mathcal{R} \models \mathcal{S}'$. The sets of constraints $\mathcal{S}$ and $\mathcal{S}'$ are *equivalent*, denoted by $\mathcal{S} \equiv \mathcal{S}'$, if $\mathcal{S} \models \mathcal{S}'$ and $\mathcal{S}' \models \mathcal{S}$. In the above, whenever $\mathcal{S} = \{C\}$ and/or $\mathcal{S}' = \{C'\}$ are singletons, we usually write C and/or C' instead of $\{C\}$ and/or $\{C'\}$.

An inference rule can be described using a schema of the form

$$\frac{\mathcal{S}}{C} \ \textit{conditions},$$

which should be informally read as "if $\mathcal{S}$, subject to some *conditions*, then C", where $\mathcal{S}$ is a set of constraints and C is a constraint. The inference rule is *k-ary* if $|\mathcal{S}| = k$. A set of inference rules $\mathfrak{R}$ is *k-ary* if every rule $r \in \mathfrak{R}$ is at most k-ary.

If a constraint C can be obtained from a set of constraints $\mathcal{S}$ by repeatedly applying rules of a set of inference rules $\mathfrak{R}$, we say that C can be derived from $\mathcal{S}$ using $\mathfrak{R}$, denoted by $\mathcal{S} \vdash_{\mathfrak{R}} C$. We usually omit $\mathfrak{R}$ if $\mathfrak{R}$ is clear from the context. A set of inference rules $\mathfrak{R}$ is *sound* whenever $\mathcal{S} \vdash_{\mathfrak{R}} C$ implies $\mathcal{S} \models C$ and *complete* whenever $\mathcal{S} \models C$ implies $\mathcal{S} \vdash_{\mathfrak{R}} C$.

A set of inference rules is a *finite axiomatization* if it is sound, complete, k-ary (for some $k \geq 0$), and if the inference rules can be represented by a finite number of decidable inference rule schemas. An inference rule schema $s =$ "if $\mathcal{S}$, subject to some *conditions*, then C" is decidable if, given a set of constraints

[1] Observe that the semantics of satisfaction depends on the type of constraints considered. The semantics of satisfaction for functional constraints is defined in Definition 5 and the semantics of satisfaction for constant constraints is defined in Definition 7. However, we use a generic notation for satisfaction independent of the type of constraints, which is why we introduce it here.

$\mathcal{S}'$ with $|\mathcal{S}| \leq |\mathcal{S}'|$ and constraint C', it is decidable if s can be applied to derive $\mathcal{S}' \vdash_s C'$.

3 Functional and constant constraints

Functional constraints and constant constraints on n-ary relations are special subclasses of equality-generating constraints on n-ary relations. We formally define functional, constant, and equality-generating constraints on n-ary relations.

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

Let $C = (P, L \to R)$ be a functional constraint. We say that P, L, and R are the pattern, left-hand side, and right-hand side of C, respectively. We denote these by $\textsc{ptn}(C)$, $\textsc{lhs}(C)$, and $\textsc{rhs}(C)$.

Definition 2 A *constant constraint* is a pair (P, E), where P is a nonempty pattern and E is a finite set of equalities of the form $(c = \$v)$ for $c \in \mathcal{U}$ and $\$v \in \mathcal{V}_P$.

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

Let $C = (P, E)$ be a constant or an equality-generating constraint. We say that P and E are the pattern and the equalities of C, respectively. We denote these by $\textsc{ptn}(C)$ and $\textsc{eq}(C)$. We say that an equality $(t = u) \in E$ is a *constant equality* on pattern P if $t \in \mathcal{U}$ and $u \in \mathcal{V}_P$. Notice that the constant constraints are a proper syntactical subclass of the equality-generating constraints, in which the set of equalities is restricted to constant equalities.

Definition 4 A *constant-functional constraint* is a constraint that is either a constant constraint or a functional constraint.

We formally define the semantics of the functional, constant, and equality-generating constraints on n-ary relations.

Definition 5 Let $C = (P, L \to R)$ be a functional constraint and let P' be a pattern. Then P' *satisfies* C if, for every pair of embeddings e_1 and e_2 of P into P' with $e_1 =_L e_2$, we have $e_1 =_R e_2$.

Each time we have two embeddings of P into P', it makes sense to look at the structure $e_1(P) \cup e_2(P)$ in order to compare the two embeddings. To make reasoning about such structures easier, we formalize them.

Definition 6 Let f_1 and f_2 be a pair of embeddings of pattern P into some pattern P' with $f_1 =_L f_2$ for $L \subseteq \mathcal{V}_P$. We say that the pattern $D = f_1(P) \cup f_2(P)$ is a *double pattern* of P and L. A pattern D is a *maximally double pattern* of P and L if f_1 and f_2 are injections, $f_1 =_{L \cup \mathrm{id}_\mathcal{U}} f_2$, $\mathrm{range}(f_1|_{\mathcal{V}_P}) \subseteq \mathcal{V}$, $\mathrm{range}(f_2|_{\mathcal{V}_P}) \subseteq \mathcal{V}$, and $\mathrm{range}(f_1|_{\mathcal{V}_P \setminus L}) \cap \mathrm{range}(f_2|_{\mathcal{V}_P \setminus L}) = \emptyset$.

We observe that if $f_1(P) \cup f_2(P)$ is a maximally double pattern, then, by construction, f_1 and f_2 are bijective functions of $\mathcal{V}_P \cup \mathcal{U}$ into $\mathcal{V}_{f_1(P)} \cup \mathcal{U}$ and $\mathcal{V}_{f_2(P)} \cup \mathcal{U}$, embedding P into $f_1(P)$ and $f_2(P)$, respectively. In addition, f_1 and f_2 map constants to themselves and variables onto variables. Hence, P, $f_1(P)$, and $f_2(P)$ are isomorphic and, as a direct consequence, the maximally double pattern of P and L is unique up to isomorphisms. We shall therefore often leave the embeddings f_1 and f_2 that define $f_1(P) \cup f_2(P)$ implicit.

Definition 7 Let P' be a pattern and let $C = (P, E)$ be a constant constraint or an equality-generating constraint. Then P' *satisfies* C if, for every embedding e of P into P' and every equality $(t = u) \in E$, we have $e(t) = e(u)$.

Akhtar et al. [3] already showed that every functional constraint can be written as an equality-generating constraint. To establish a formal relationship between equality-generating and functional constraints, we provide the following results.

Proposition 1 *Let $C = (P, L \to R)$ and let $D = f_1(P) \cup f_2(P)$ be the maximally double pattern of P and L. For every two embeddings e_1 and e_2 of P into pattern P' with $e_1 =_L e_2$, there is an embedding e of D into P' with, for every $\$r \in R$, $e_1(\$r) = e \circ f_1(\$r)$ and $e_2(\$r) = e \circ f_2(\$r)$. Conversely, for every embedding e of D into P', there are two embeddings e_1 and e_2 of P into P' with $e_1 =_L e_2$ and, for every $\$r \in R$, $e \circ f_1(\$r) = e_1(\$r)$ and $e \circ f_2(\$r) = e_2(\$r)$.*

Proof In one direction, we define $e = ({e_1} \circ {f_1}^{-1}) \cup ({e_2} \circ {f_2}^{-1})$, and, in the other direction, we define $e_1 = e \circ f_1$ and $e_2 = e \circ f_2$. □

Corollary 1 *Let $C = (P, L \to R)$ be a functional constraint and let $D = f_1(P) \cup f_2(P)$ be the maximally double pattern of P and L. We have $C \equiv (D, \{(f_1(\$r) = f_2(\$r)) \mid \$r \in R\})$.*

As already mentioned, the functional constraints are a generalization of the functional dependencies. We also formalize this relationship.

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 $C' = (\{(A_1, \ldots, A_n)\}, L \to R)$, in which the attribute names are assumed to be variables. Then $C \equiv C'$.*

The functional dependencies have a well-known axiomatization in the form of *Armstrong's axioms*, consisting of the three inference rules *Reflexivity*, *Augmentation*, and *Transitivity* [5]. We generalize Armstrong's inference rules to our setting of the functional constraints.

Rule 1 (Reflexivity) *Let P be a pattern. If $R \subseteq L \subseteq \mathcal{V}_P$, then $(P, L \to R)$.*

Rule 2 (Augmentation) *If $(P, L \to R)$ and $V \subseteq \mathcal{V}_P$, then $(P, L \cup V \to R \cup V)$.*

Rule 3 (Transitivity) *If $(P, L \to M)$ and $(P, M \to R)$, then $(P, L \to R)$.*

Since Armstrong's axioms can be generalized to functional constraints, it is straightforward to show that the well-known decomposition and union rules can also be generalized to functional constraints. Also, similar decomposition and union rules can be obtained for the equality-generating constraints [3]. For the decomposition and union of constant constraints, we introduce the following inference rules, which also hold for the equality-generating constraints in general.

Rule 4 (Decomposition) *If* $(P, E \cup E')$*, then* (P, E)*.*

Rule 5 (Union) *If* (P, E) *and* (P, E')*, then* $(P, E \cup E')$*.*

From now on, if the left-hand side or right-hand side in a functional constraint is a singleton set $\{\$v\}$, then we usually write $\$v$ instead. Likewise, if the set of equalities in a equality-generating or constant constraint is a singleton set $\{(t = u)\}$, then we usually write $(t = u)$ instead.

Armstrong's axioms together with Decomposition and Union are not complete for the functional constraints, the constant constraints, or the constant-functional constraints: these inference rules only provide means to reason on constraints specified on a single pattern. The following example exhibits a situation in which this is not sufficient:

Example 4 Consider the functional constraint $C = (\{(\$a, \$b)\}, \$a \to \$b)$ and the pattern $P' = \{(\$a, c_1), (\$a, c_2)\}$ with $c_1, c_2 \in \mathcal{U}$ and $c_1 \neq c_2$. If C holds on a relation $\mathcal{R}$, then no embedding of P' into $\mathcal{R}$ is possible, and, hence, every constant-functional constraint on the pattern P' holds.

Likewise, consider the constant constraint $C = (\{(\$a, \$b)\}, (c = \$b))$ and the pattern $P' = \{(\$a, c')\}$ with $c' \in \mathcal{U}$ and $c \neq c'$. Also in this case, if C holds on a relation $\mathcal{R}$, then no embedding of P' into $\mathcal{R}$ is possible, and, hence, every constant-functional constraint on the pattern P' holds.

Both these cases consider the derivation of a constraint on pattern P' from a constraint C on pattern $\textsc{ptn}(C)$ with $P' \neq \textsc{ptn}(C)$. None of the presented inference rules are, however, able to reason about constraints specified on patterns that are not all equivalent.

4 Chasing constant-functional constraints

For equality-generating constraints, Algorithm 1 is known to decide implication [3]. This algorithm is a constant-aware variation of standard chase algorithms for equality-generating dependencies [2,8].

In Algorithm 1, we refer to lines 5–10 as *equalization steps*, to line 12 as *inconsistency termination*, and to line 15 as *regular termination*.

Theorem 1 (Akhtar et al. [3]) *Algorithm 1 is correct for equality-generating constraints.*

We use the relationship between the constant-functional constraints and the equality-generating constraints, as described in Corollary 1, to construct a chase-based algorithm that decides implication of constant-functional constraints, shown as Algorithm 2.

In Algorithm 2, we refer to lines 15–21 as *equalization steps*, to line 23 as *inconsistency termination*, and to line 26 as *regular termination*. This is analogue to the equalization steps, inconsistency termination, and regular termination in Algorithm 1.

Theorem 2 *Algorithm 2 is correct for constant-functional constraints.*

Algorithm 1 Chase for equality-generating constraints

Input: Set of equality-generating constraints $\mathcal{S}$,
Equality-generating constraint C
Output: $\mathcal{S} \models C$

$\mathfrak{T} \leftarrow \text{PTN}(C)$
while there exists a constraint $C' \in \mathcal{S}$ with $\mathfrak{T} \not\models C'$ **do**
  Choose equality $(t' = u') \in \text{EQ}(C')$ and
    embedding e of $\text{PTN}(C')$ into $\mathfrak{T}$ with $e(t') \neq e(u')$
  /* *equalize* $e(t')$ *and* $e(u')$ *in* $\mathfrak{T}$ */
  if $e(t') \in \mathcal{V}$ **then**
    /* *replace all occurrences of* $e(t')$ *in* $\mathfrak{T}$ *by* $e(u')$ */
    $\mathfrak{T} \leftarrow \phi_{e(u') \hookleftarrow e(t')}(\mathfrak{T})$
  else if $e(u') \in \mathcal{V}$ **then**
    /* *replace all occurrences of* $e(u')$ *in* $\mathfrak{T}$ *by* $e(t')$ */
    $\mathfrak{T} \leftarrow \phi_{e(t') \hookleftarrow e(u')}(\mathfrak{T})$
  else /* $e(t'), e(u') \in \mathcal{U}$ *and* $e(t') \neq e(u')$ */
    return TRUE
  end if
end while
return $\mathfrak{T} \models C$

Algorithm 2 Chase for constant-functional constraints

Input: Set of constant-functional constraints $\mathcal{S}$,
Constant-functional constraint C
Output: $\mathcal{S} \models C$

if C is a functional constraint **then**
  Let D be the maximally double pattern of $\text{PTN}(C)$ and $\text{LHS}(C)$
  $\mathfrak{T} \leftarrow D$
else /* C *is a constant constraint* */
  $\mathfrak{T} \leftarrow \text{PTN}(C)$
end if
while there exists a constraint $C' \in \mathcal{S}$ with $\mathfrak{T} \not\models C'$ **do**
  if C' is a functional constraint **then**
    Choose variable $\$r' \in \text{RHS}(C')$ and pair of embeddings e_1 and e_2 of
      $\text{PTN}(C)$ into $\mathfrak{T}$ with $e_1 =_{\text{LHS}(C')} e_2$ and $e_1(\$r') \neq e_2(\$r')$
    $t_1, t_2 \leftarrow e_1(\$r'), e_2(\$r')$
  else /* C' *is a constant constraint* */
    Choose equality $(c = \$v) \in \text{EQ}(C')$ and
      embedding e of $\text{PTN}(C')$ into $\mathfrak{T}$ with $c \neq e(\$v)$
    $t_1, t_2 \leftarrow c, e(\$v)$
  end if
  /* *equalize* t_1 *and* t_2 *in* $\mathfrak{T}$ */
  if $t_1 \in \mathcal{V}$ **then**
    /* *replace all occurrences of* t_1 *in* $\mathfrak{T}$ *by* t_2 */
    $\mathfrak{T} \leftarrow \phi_{t_2 \hookleftarrow t_1}(\mathfrak{T})$
  else if $t_2 \in \mathcal{V}$ **then**
    /* *replace all occurrences of* t_2 *in* $\mathfrak{T}$ *by* t_1 */
    $\mathfrak{T} \leftarrow \phi_{t_1 \hookleftarrow t_2}(\mathfrak{T})$
  else /* $t_1, t_2 \in \mathcal{U}$ *and* $t_1 \neq t_2$ */
    return TRUE
  end if
end while
return $\mathfrak{T} \models C$

Proof Algorithm 2 makes a case distinction on the type of all constraints involved. In it, constant constraints are treated as normal equality-generating con-

straints. For functional constraints, the algorithm implicitly translates functional constraints to equality-generating constraints using the results of Proposition 1 and Corollary 1. □

In Section 5, we shall use the correctness of Algorithm 2 to prove that there is a complete axiomatization for the derivation of constant-functional constraints from a set of constant-functional constraints.

5 Axiomatizing constant-functional constraints

Algorithm 2 is a correct algorithm to decide $\mathcal{S} \models C$, with $\mathcal{S}$ a set of constant-functional constraints and C a single constant-functional constraint. Hence, if $\mathcal{S} \models C$ holds, and if we can simulate every equalization step and the termination of an execution of Algorithm 2 by sound inference rules, then we have a sound and complete axiomatization for deriving constant-functional constraints.

For chases that do not perform equalization steps, we provide straightforward inference rules to simulate the eventual termination step in a single inference step (Section 5.1).

For chases that do initially perform an equalization step, we show that we can reduce any chase that decides $\mathcal{S} \models C$ to a single equalization step (constant constraints) or at most two equalization steps (functional constraints) with a single constraint $C' \in \mathcal{S}$, followed by the chase deciding $\mathcal{S} \models C''$, for some constraint C''. We do so by showing that, after the initial equalization step(s) with C', the resulting tableau $\mathfrak{T}$ is equivalent to the initial tableau for any chase deciding $\mathcal{S} \models C''$. To show that $\mathcal{S} \models C''$ holds when $\mathcal{S} \models C$ holds, we show $C \models C''$. Lastly, we show $\{C', C''\} \models C$, while providing sound inference rules to derive $\{C', C''\} \vdash C$ in a finite number of inference steps.

Due to the dual nature of Algorithm 2 with respect to, on the one hand, constant constraints, and on the other hand, functional constraints, we divide our search for inference rules to simulate the initial equalization step(s) of an execution of Algorithm 2 into two cases.

Firstly, we consider the derivation of constant constraints (Section 5.2). Thereto, we simulate the initial equalization step in the chase that decides $\mathcal{S} \models C$ with C a constant constraint by using a finite number of inference steps.

Secondly, we consider the derivation of functional constraints (Section 5.3). Thereto we try to simulate the chase that decides $\mathcal{S} \models C$ with C a functional constraint. In this case, we conclude that a direct simulation of individual equalization steps is not straightforward. To circumvent this problem, we show that, in this case, Algorithm 2 can be normalized to a more specialized, symmetry-preserving, chase algorithm. For the symmetry-preserving chase that decides $\mathcal{S} \models C$, we are able to simulate the initial symmetry-preserving step by using a finite number of inference steps.

Finally, the simulation of direct termination in chases for constant-functional constraints, the initial equalization step in chases for constant constraints, and the initial symmetry-preserving step in chases for functional constraints are used as the basis for an induction argument to extend the simulation to the entire chase (Section 5.4). This induction argument shows how to construct a derivation $\mathcal{S} \vdash C$ by showing how to translate each equalization step (constant constraints) or

symmetry-preserving step (functional constraints) of a chase deciding $\mathcal{S} \models C$ to a finite number of inference steps, this such that the produced sequence of inference steps is a derivation of $\mathcal{S} \vdash C$.

5.1 Inference rules for direct termination

Consider the case where Algorithm 2 decides whether $\mathcal{S} \models C$ terminates without performing any equalization steps. Two subcases are possible, namely regular termination and inconsistency termination.

5.1.1 Regular termination

Assuming no equalization steps are possible, we have regular termination if there does not exist a constraint $C' \in \mathcal{S}$ with $\mathfrak{T} \not\models C'$. In this case, due to line 26 in Algorithm 2, we have $\mathcal{S} \models C$ if and only if $\mathfrak{T} \models C$. We investigate necessary conditions on C such that $\mathfrak{T} \models C$ holds.

Proposition 3 *Let C be a functional constraint and let D be the maximally double pattern of $\textsc{ptn}(C)$ and $\textsc{lhs}(C)$. If $D \models C$, then $\textsc{rhs}(C) \subseteq \textsc{lhs}(C)$.*

If we have regular termination and C is a functional constraint, then $\mathfrak{T}$ is equivalent to the maximally double pattern of $\textsc{ptn}(C)$ and $\textsc{lhs}(C)$. Hence, by Proposition 3, we conclude $\textsc{rhs}(C) \subseteq \textsc{lhs}(C)$, and thus we can use Reflexivity to derive $\emptyset \vdash C$. Hence, also $\mathcal{S} \vdash C$.

Proposition 4 *Let C be a constant constraint. If $\textsc{ptn}(C) \models C$, then $\textsc{eq}(C) = \emptyset$.*

If we have regular termination and C is a constant constraint, then $\mathfrak{T}$ is equivalent to $\textsc{ptn}(C)$. Hence, by Proposition 4, we conclude $\textsc{eq}(C) = \emptyset$. For this case, we introduce the following inference rule.

Rule 6 (Empty) *Let P be a pattern. We have $(P, \emptyset)$.*

Proof (soundness) Let $\mathcal{R}$ be a relation and assume there are embeddings of P into $\mathcal{R}$. Let e be any embedding of P into $\mathcal{R}$. The constraint specifies no equalities, hence, every equality specified by the constraint is satisfied by embedding e. □

If we have regular termination and C is a constant constraint, then $\textsc{eq}(C) = \emptyset$, and thus we can use Empty to derive $\emptyset \vdash C$. Hence, also $\mathcal{S} \vdash C$.

5.1.2 Inconsistency termination

Still assuming no equalization steps are possible, we have inconsistency termination if there does exist a constraint $C' \in \mathcal{S}$ with $\mathfrak{T} \not\models C'$. In this case, due to line 23 in Algorithm 2, we can find two terms $t_1 \in \mathcal{U}$ and $t_2 \in \mathcal{U}$ with $t_1 \neq t_2$ that should be equal according to C. We introduce the following inference rules to deal with such inconsistencies.

Rule 7 (Inconsistency I) *Let P be a pattern. If $(P', L' \to R')$ with $\$r' \in R'$, and if there is a pair of embeddings of P' into P that agree on L' and map $\$r'$ to two distinct constants, then (P, E), for every finite set of constant equalities E on P, and $(P, L \to R)$, for every $L \subseteq \mathcal{V}_P$ and $R \subseteq \mathcal{V}_P$.*

Proof (soundness) Let $\mathcal{R}$ be a relation with $\mathcal{R} \models (P', L' \to R')$, and assume there are embeddings of P into $\mathcal{R}$. Let e be such an embedding. Let e'_1 and e'_2 be embeddings mapping P' into P that agree on L' and map $\$r'$ to two distinct constants. Now, the embeddings $e''_1 = e \circ e'_1$ and $e''_2 = e \circ e'_2$ map P' into $\mathcal{R}$, agree on L', and map $\$r'$ to two distinct constants, a contradiction. Hence, no embedding e of P into $\mathcal{R}$ exists. Thus, we can conclude $\mathcal{R} \models (P, E)$, for every finite set of constant equalities E on P, and $\mathcal{R} \models (P, L \to R)$, for every $L \subseteq \mathcal{V}_P$ and $R \subseteq \mathcal{V}_P$. □

Rule 8 (Inconsistency II) *Let P be a pattern. If (P', E') with $(c = \$v) \in E'$, and if there is an embedding of P' into P mapping $\$v$ to a constant unequal to c, then (P, E), for every finite set of constant equalities E on P, and $(P, L \to R)$, for every $L \subseteq \mathcal{V}_P$ and $R \subseteq \mathcal{V}_P$.*

Proof (soundness) Let $\mathcal{R}$ be a relation with $\mathcal{R} \models (P', E')$, and assume there are embeddings of P into $\mathcal{R}$. Let e be such an embedding. Let e' be an embedding mapping P' into P and mapping $\$v$ to a constant unequal to c. Now, the embedding $e'' = e \circ e'$ maps P' into $\mathcal{R}$ and maps $\$v$ to a constant unequal to c, a contradiction. Hence, no embedding e of P into $\mathcal{R}$ exists. Thus, we can conclude $\mathcal{R} \models (P, E)$, for every finite set of constant equalities E on P, and $\mathcal{R} \models (P, L \to R)$, for every $L \subseteq \mathcal{V}_P$ and $R \subseteq \mathcal{V}_P$. □

It is straightforward to verify that these inference rules can be applied to simulate chases that decide $\mathcal{S} \models C$, with C a constant constraint, with inconsistency termination and without using equalization steps. We shall now prove that this is also the case when C is a functional constraint.

Proposition 5 *Let $\mathcal{S} \models C$ with C a functional constraint. If Algorithm 2 decides $\mathcal{S} \models C$ by immediate inconsistency termination using constraint $C' \in \mathcal{S}$, then Rule 7 or Rule 8 can be applied to derive $C' \vdash C$.*

Proof Let $C = (P, L \to R)$ and let $\mathfrak{T} = D = f_1(P) \cup f_2(P)$ be the maximally double pattern of P and L. Let e be any embedding of $\text{PTN}(C')$ into D. As P, $f_1(P)$, and $f_2(P)$ are isomorphic and f_1 and f_2 are bijections, the function $e' = \Phi_{f_1 \hookleftarrow f_2} \circ e$ is well-defined and an embedding of $\text{PTN}(C')$ into $f_1(P)$. Hence, the function $e'' = {f_1}^{-1} \circ e' = {f_1}^{-1} \circ \Phi_{f_1 \hookleftarrow f_2} \circ e$ is also well-defined and an embedding of $\text{PTN}(C')$ into P. By construction, the function ${f_1}^{-1} \circ \Phi_{f_1 \hookleftarrow f_2}$ is the identity on constants. We now consider two cases, namely where C' is a functional constraint and where C' is a constant constraint.

1. If $C' = (P', L' \to R')$ is a functional constraint, then, since Algorithm 2 must pass line 9 to terminate on line 23, we can choose a variable $\$r' \in R'$ and pair of embeddings e_1 and e_2 of P' into $\mathfrak{T} = D$ with $e_1 =_{L'} e_2$ and $e_1(\$r') \neq e_2(\$r')$. Now, using the above construction, we obtain the pair of embeddings $e''_1 = {f_1}^{-1} \circ \Phi_{f_1 \hookleftarrow f_2} \circ e_1$ and $e''_2 = {f_1}^{-1} \circ \Phi_{f_1 \hookleftarrow f_2} \circ e_2$ of P' into P. As $e_1 =_{L'} e_2$, we also have $e''_1 =_{L'} e''_2$. Furthermore, as $e_1(\$r') \in \mathcal{U}$, $e_2(\$r') \in \mathcal{U}$, $e_1(\$r') \neq e_2(\$r')$, and ${f_1}^{-1} \circ \Phi_{f_1 \hookleftarrow f_2}$ is the identity on constants, we have $e''_1(\$r') \in \mathcal{U}, e''_2(\$r') \in \mathcal{U}$, and $e''_1(\$r') \neq e''_2(\$r')$. We can thus apply Inconsistency I using constraint C' and embeddings e''_1 and e''_2 to derive $C' \vdash C$.

2. If $C' = (P', E')$ is a constant constraint, then, since Algorithm 2 must pass line 12 to terminate on line 23, we can choose an equality $(c = \$v) \in E'$ and embedding e of P' into $\mathfrak{T} = D$ with $c \neq e(\$v)$. Now, using the above construction, we obtain an embedding $e'' = {f_1}^{-1} \circ \Phi_{f_1 \leftarrow\!\!\shortmid f_2} \circ e$ of P' into P. As $e(\$v) \in \mathcal{U}$ and ${f_1}^{-1} \circ \Phi_{f_1 \leftarrow\!\!\shortmid f_2}$ is the identity on constants, we have $e''(\$v) \in \mathcal{U}$ and $c = e''(c) \neq e''(\$v)$. We can thus apply Inconsistency II using constraint C' and embedding e'' to derive $C' \vdash C$.

This case analysis completes the proof. □

5.2 Inference rules for deriving constant constraints

From now on, we assume that a chase deciding $\mathcal{S} \models C$ always performs equalization steps. For simulating the initial equalization step in the case where C is a constant constraint, we use the following three inference rules.

Rule 9 (Application I) *Let P be a pattern, let $\$v \in \mathcal{V}_P$ be a variable, and let $t \in \mathcal{V}_P \cup \mathcal{U}$ be a term. If $(\phi_{t \leftarrow\!\!\shortmid \$v}(P), E)$, if $(P', L' \to R')$, and if there is a pair of embeddings of P' into P that agree on L' and map $\$r' \in R'$ to t and $\$v$, respectively, then (P, E). If, in addition, there exists a $c \in \mathcal{U}$ such that either $c = t$ (if $t \in \mathcal{U}$) or $(c = t) \in E$ (if $t \in \mathcal{V}$), then also $(P, E \cup \{(c = \$v)\})$.*

Proof (soundness) Let $\mathcal{R}$ be a relation with $\mathcal{R} \models (\phi_{t \leftarrow\!\!\shortmid \$v}(P), E)$ and $\mathcal{R} \models (P', L' \to R')$, and assume there are embeddings of P into $\mathcal{R}$. Let e be any embedding of P into $\mathcal{R}$ and let e'_1 and e'_2 be any pair of embeddings of P' into P that agree on L' and map $\$r' \in R'$ to t and $\$v$, respectively. Now $e''_1 = e \circ e'_1$ and $e''_2 = e \circ e'_2$ are embeddings of P' into $\mathcal{R}$ with $e''_1 =_{L'} e''_2$. By $\mathcal{R} \models (P', L' \to R')$, we have $e''_1(\$r') = e''_2(\$r')$, and, by construction, we have $e''_1(\$r') = e(t)$ and $e''_2(\$r') = e(\$v)$. Hence, $e(\$v) = e(t)$, and $\varepsilon = e|_{\mathrm{domain}(e) \setminus \{\$v\}}$ is an embedding of $\phi_{t \leftarrow\!\!\shortmid \$v}(P)$ into $\mathcal{R}$. Now, by $\mathcal{R} \models (\phi_{t \leftarrow\!\!\shortmid \$v}(P), E)$, we have, for every $(c' = \$w) \in E$, $\varepsilon(\$w) = c'$. Notice that $\$v \notin \mathrm{domain}(\varepsilon)$. Hence, if $\varepsilon(\$w) = c'$, then $\$w \neq \$v$ and $e(\$w) = c'$. Hence, we conclude $\mathcal{R} \models (P, E)$.

Above, we already showed that $e(\$v) = e(t)$. It now follows readily that, if, in addition, there exists $c \in \mathcal{U}$ such that either $c = t$ (if $t \in \mathcal{U}$) or $(c = t) \in E$ (if $t \in \mathcal{V}$), we also have $\mathcal{R} \models (P, (c = \$v))$, and, hence, $\mathcal{R} \models (P, E \cup \{(c = \$v)\})$. □

Rule 10 (Application II) *Let P be a pattern, let $\$v \in \mathcal{V}_P$ be a variable, and let $c \in \mathcal{U}$ be a constant. If $(\phi_{c \leftarrow\!\!\shortmid \$v}(P), E)$, if (P', E') with $(c = \$v') \in E'$, and if there is an embedding of P' into P mapping $\$v'$ to $\$v$, then $(P, E \cup \{(c = \$v)\})$.*

Proof (soundness) Let $\mathcal{R}$ be a relation with $\mathcal{R} \models (\phi_{c \leftarrow\!\!\shortmid \$v}(P), E)$ and $\mathcal{R} \models (P', E')$, and assume there are embeddings of P into $\mathcal{R}$. Let e be any embedding of P into $\mathcal{R}$ and let e' be any embedding of P' into P mapping $\$v'$ to $\$v$. Now $e'' = e \circ e'$ is an embedding of P' into $\mathcal{R}$. By $\mathcal{R} \models (P', E')$, we have $e''(\$v') = e(\$v) = c$.

As a consequence, $\varepsilon = e|_{\mathrm{domain}(e) \setminus \{\$v\}}$ is an embedding of $\phi_{c \leftarrow\!\!\shortmid \$v}(P)$ into $\mathcal{R}$. Now, by $\mathcal{R} \models (\phi_{t \leftarrow\!\!\shortmid \$v}(P), E)$, we have, for every $(c' = \$w) \in E$, $\varepsilon(\$w) = c'$. Notice that $\$v \notin \mathrm{domain}(\varepsilon)$. Hence, if $\varepsilon(\$w) = c'$, then $\$w \neq \$v$ and $e(\$w) = c'$. Hence, we conclude $\mathcal{R} \models (P, E \cup \{(c = \$v)\})$. □

The following example exhibits situations in which the Application I and II rules can be used.

Example 5 Using Application I in a straightforward manner, we can derive

$$\{(\{(\$a, \$b, \$c)\}, \$a \to \$b), (\{(\$a, b, \$c), (\$a, b, d)\}, (c = \$c))\} \vdash \\ (\{(\$a, b, \$c), (\$a, \$b, d)\}, (c = \$c))$$

by using the embeddings

$$e_1 = \{\$a \mapsto \$a, \$b \mapsto b, \$c \mapsto \$c\}$$
$$e_2 = \{\$a \mapsto \$a, \$b \mapsto \$b, \$c \mapsto d\}.$$

We thus have $e_1 =_{\$a} e_2$, $e_1(\$b) = b$, and $e_2(\$b) = \$b$. Using Application I, we can also derive

$$(\{(\$a, \$b, \$c), \$a \to \$c) \vdash (\{(\$a, \$b, c), (\$a, \$b, \$c)\}, \{c = \$c\})$$

by using the embeddings

$$e_1 = \{\$a \mapsto \$a, \$b \mapsto \$b, \$c \mapsto c\}$$
$$e_2 = \{\$a \mapsto \$a, \$b \mapsto \$b, \$c \mapsto \$c\}.$$

Observe that in this case, Application I is applied to a single non-trivial functional constraint and to the trivial constant constraint $(\{(\$a, \$b, c), (\$a, \$b, \$c)\}, \emptyset)$. Using Application II, we can derive

$$\{(\{(\$a, \$b, \$c)\}, (c = \$c)), (\{(\$a, \$b, c)\}, (b = \$b))\} \vdash (\{(\$a, \$b, \$c)\}, (b = \$b))$$

by using the embedding $e' = \{\$a \mapsto \$a, \$b \mapsto \$b, \$c \mapsto \$c\}$.

Rule 11 (Inconsistent Propagation) *Let P be a pattern, $\$v \in \mathcal{V}_P$, and $c_1, c_2 \in \mathcal{U}$ with $c_1 \neq c_2$. If $(P, \{(c_1 = \$v), (c_2 = \$v)\})$, then (P, E) for every finite set of constant equalities E on P.*

Proof (soundness) Let $\mathcal{R}$ be a relation with $\mathcal{R} \models (P, \{(c_1 = \$v), (c_2 = \$v)\})$ and assume there are embeddings of P into $\mathcal{R}$. Let e be such an embedding mapping P into $\mathcal{R}$. By $\mathcal{R} \models (P, \{(c_1 = \$v), (c_2 = \$v)\})$, we must have $c_1 = e(\$v)$ and $c_2 = e(\$v)$. As a consequence, we conclude $c_1 = c_2$, a contradiction. Hence, no embedding e of P into $\mathcal{R}$ exists. Thus we can conclude $\mathcal{R} \models (P, E)$, for every finite set of constant equalities E on P. □

Before we show how the initial equalization step in any chase deciding $\mathcal{S} \models C$ with C a constant constraint can be simulated by the introduced inference rules, we introduce an additional inference rule to simplify our proofs:

Rule 12 (Embedding I) *If (P', E') and h is an embedding from P' into P, then $(P, \{(c = h(\$v)) \mid ((c = \$v) \in E') \wedge (h(\$v) \in \mathcal{V})\})$.*

Proof (soundness) Let $\mathcal{R}$ be a relation with $\mathcal{R} \models (P', E')$, and assume there are embeddings of P into $\mathcal{R}$. Let e be any embedding of P into $\mathcal{R}$. Then $\varepsilon = e \circ h$ is an embedding of P' into $\mathcal{R}$. Hence, for every $(c = \$v) \in E'$, we have $e(c) = c = \varepsilon(c) = \varepsilon(\$v) = e(h(\$v))$. □

Embedding I explicitly maps a constant constraint defined on a pattern to a different pattern. Observe that Embedding I is applicable to equality-generating constraints as well. However, if we apply Embedding I to a constant constraint (P', E'), then the derived constraint is always a constant constraint. The following example illustrates the usage of Embedding I.

Example 6 If $(\{(\$a, \$b)\}, (\$a = c))$ holds, then trivially also $(\{(\$x, \$y)\}, (\$x = c))$ holds. We can derive $(\{(\$x, \$y)\}, (\$x = c))$ from $(\{(\$a, \$b)\}, (\$a = c))$ by using Embedding I with the embedding $h = \{\$a \mapsto \$x, \$b \mapsto \$y\}$.

Next, we show how the initial equalization step in any chase deciding $\mathcal{S} \models C$ with C a constant constraint can be simulated by the introduced inference rules.

Theorem 3 *Let $\mathcal{S}$ be a set of constant-functional constraints, let $C = (P, E)$ be a constant constraint, and let $\mathcal{S} \models C$.*

If Algorithm 2 is initially able to perform an equalization step with $C' \in \mathcal{S}$ resulting in tableau $\mathfrak{T}'' = P''$, then there exists a constant constraint $C'' = (P'', E'')$ such that we have the following:

1. $\mathcal{S} \models C''$,
2. $\{C', C''\} \vdash C$ *using Rules 4, 9, 10, and 11.*

Proof Without loss of generality, we can assume that $\phi_{t_1 \leftrightarrow t_2}$ is the equalization performed in the initial equalization step: this always holds if C' is a constant constraint, and, if C' is a functional constraint, then we can always swap the roles of e_1 and e_2. Hence, we assume that $P'' = \phi_{t_1 \leftrightarrow t_2}(P)$.

We distinguish two types of executions of Algorithm 2, namely those that terminate regularly and those that terminate due to inconsistency. We divide our analysis into these two cases.

1. Algorithm 2 terminates regularly. We define

$$E'' = \{c = \phi_{t_1 \leftrightarrow t_2}(\$v) \mid ((c = \$v) \in E) \wedge (\phi_{t_1 \leftrightarrow t_2}(\$v) \in \mathcal{V})\}$$

and $C'' = (P'', E'')$. Observe that we have $E \neq E''$ if and only if there exists an equality $(c = t_2) \in E$: if $(c = t_2) \in E$ and $t_1 \in \mathcal{V}$, then we have $E'' = E \cup \{(c = t_1)\} \setminus \{(c = t_2)\}$, and, if $(c = t_2) \in E$ and $t_1 \in \mathcal{U}$, then we have $E'' = E \setminus \{(c = t_2)\}$. Also observe that, if $t_1 \in \mathcal{U}$ and if there exists an equality $(c = t_2) \in E$, then, due to this chase terminating regularly and $\mathcal{S} \models C$, we have $t_1 = c$.

We have $C \models C''$, since we have $C \vdash C''$ by a straightforward application of Embedding I using the embedding $\phi_{t_1 \leftrightarrow t_2}$. By $\mathcal{S} \models C$ and $C \models C''$, we conclude $\mathcal{S} \models C''$. Next, we prove $\{C', C''\} \vdash C$. We do so by a case distinction on the type of constraint C'.

1.a. C' is a functional constraint. Let e_1 and e_2 be embeddings of $\text{PTN}(C')$ into $\mathfrak{T} = P$ and let $\$r' \in \text{RHS}(C')$ with $t_1 = e_1(\$r')$ and $t_2 = e_2(\$r')$, meeting the conditions of line 9 of Algorithm 2. Since Algorithm 2 terminates regularly, it is not possible that $t_1, t_2 \in \mathcal{U}$, and, since $\phi_{t_1 \leftrightarrow t_2}$ is the equalization performed, it is not possible that $t_1 \in \mathcal{V}, t_2 \in \mathcal{U}$. Two cases remain, namely $t_1, t_2 \in \mathcal{V}$ and $t_1 \in \mathcal{U}, t_2 \in \mathcal{V}$.

First consider the case where $t_1 \in \mathcal{V}$ and $t_2 \in \mathcal{V}$. If there exists an equality $(c = t_2) \in E$, then we have $E'' = E \cup \{(c = t_1)\} \setminus \{(c = t_2)\}$, and, if there does

not exist an equality $(c = t_2) \in E$, then we have $E'' = E$. In both cases, we apply Application I to conclude $\{C', C''\} \vdash C$. Next consider the case where $t_1 \in \mathcal{U}$ and $t_2 \in \mathcal{V}$. In this case, we have $E'' = E \setminus \{(t_1 = t_2)\}$ and we apply Application I to conclude $\{C', C''\} \vdash (P, E'' \cup \{(t_1 = t_2)\})$ and Decomposition to conclude $\{C', C''\} \vdash (P, E'')$. If $(t_1 = t_2) \in E$, then $C = (P, E'' \cup \{(t_1 = t_2)\})$, otherwise $C = (P, E'')$. Hence, we conclude $\{C', C''\} \vdash C$.

1.b. C' is a constant constraint. Let e be an embedding of $\text{PTN}(C')$ into $\mathfrak{T} = P$ and let $(c = \$v) \in \text{EQ}(C')$ be an equality with $t_1 = c$ and $t_2 = e(\$v)$, meeting the conditions of line 12 of Algorithm 2. In this case, we have $E'' = E \setminus \{(t_1 = t_2)\}$. We apply Application II to conclude $\{C', C''\} \vdash (P, E'' \cup \{(t_1 = t_2)\})$ and Decomposition to conclude $\{C', C''\} \vdash (P, E'')$. If $(t_1 = t_2) \in E$, then $C = (P, E'' \cup \{(t_1 = t_2)\})$, otherwise $C = (P, E'')$. Hence, we conclude $\{C', C''\} \vdash C$.

2. Algorithm 2 terminates due to inconsistency. Let $\$w \in \mathcal{V}_{P''}$ be a variable and let $c_1, c_2 \in \mathcal{U}$ be constants with $c_1 \neq c_2$. We define $E'' = \{(c_1 = \$w), (c_2 = \$w)\}$ and $C'' = (P'', E'')$. Chasing pattern P'' using $\mathcal{S}$ will lead to inconsistency, hence, $\mathcal{S} \models C''$. Next, we prove $\{C', C''\} \vdash C$. We do so by a case distinction on the type of constraint C'.

2.a. C' is a functional constraint. Let e_1 and e_2 be embeddings of $\text{PTN}(C')$ into $\mathfrak{T} = P$ and let $\$r' \in \text{RHS}(C')$ with $t_1 = e_1(\$r')$ and $t_2 = e_2(\$r')$, meeting the conditions of line 9 of Algorithm 2. We apply Application I to conclude (P, E'') and Inconsistent Propagation to conclude $\{C', C''\} \vdash C$.

2.b. C' is a constant constraint. Let e be an embedding of $\text{PTN}(C')$ into $\mathfrak{T} = P$ and let $(c = \$v) \in \text{EQ}(C')$ be an equality with $t_1 = c$ and $t_2 = e(\$v)$, meeting the conditions of line 12 of Algorithm 2. We apply Application II to conclude $\{C', C''\} \vdash (P, E'' \cup \{(c = \$v)\})$, Decomposition to conclude (P, E''), and Inconsistent Propagation to conclude $\{C', C''\} \vdash C$.

This case analysis completes the proof. □

5.3 Inference rules for deriving functional constraints

For chases deciding $\mathcal{S} \models C$ with C a constant constraint, there is a direct correspondence between the tableau $\mathfrak{T}$ and patterns of constant constraints. In Section 5.2, we showed that we can use this correspondence to derive inference rules simulating the initial equalization step.

For chases deciding $\mathcal{S} \models C$ with $C = (P, L \to R)$ a functional constraint, such a direct correspondence between the tableau $\mathfrak{T}$ and patterns of functional constraints does not exist. Line 3 of Algorithm 2 does however give an initial correspondence between the tableau $\mathfrak{T}$ and the maximally double pattern of P and L. A single equalization step can however destroy any correspondence between the tableau $\mathfrak{T}$ and any maximally double pattern that is useful in derivations of non-trivial functional constraints, as shown by the next example.

Example 7 We apply Algorithm 2 to the set of constraints

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

and the target functional constraint $C = (\{(\$a, \$b, \$c), (\$a, \$b, e)\}, \$a \to \$b)$. It is straightforward to verify $\mathcal{S} \models C$. We initially have the tableau

$$\mathfrak{T} = \{(\$a, \$b_1, \$c_1), (\$a, \$b_1, e), (\$a, \$b_2, \$c_2), (\$a, \$b_2, e)\}\,.$$

We have $(\{(\$a, \$b, \$c)\}, \$b \to \$c) \not\models \mathfrak{T}$, leading to the equalization $\phi_{e \leftrightarrow \$c_1}(\mathfrak{T})$ which results in the tableau

$$\mathfrak{T}' = \{(\$a, \$b_1, e), (\$a, \$b_2, \$c_2), (\$a, \$b_2, e)\}\,.$$

Using the definition of a maximally double pattern, we can search for a pattern P', set of variables $L' \subseteq \mathcal{V}_{P'}$, and injective functions f_1 and f_2 such that $\mathfrak{T}' = f_1(P') \cup f_2(P')$, $f_1 =_{L' \cup \mathrm{id}_{\mathcal{U}}} f_2$, and $\mathrm{range}(f_1|_{\mathcal{V}_{P'} \setminus L'}) \cap \mathrm{range}(f_2|_{\mathcal{V}_{P'} \setminus L'}) = \emptyset$. It is easily verifiable that the only way to achieve this is by putting $P' = \mathfrak{T}'$, $f_1 = f_2 = \mathrm{id}_{\mathcal{V}_{P'} \cup \mathcal{U}}$, and $L' = \{\$a, \$b_1, \$b_2, \$c_2\}$. Since $L' = \mathcal{V}_{P'}$, any functional constraint of the form $(P', L' \to R')$ will have $R' \subseteq L'$, and, hence, holds trivially.

We can however always maintain a direct correspondence between tableau $\mathfrak{T}$ and a maximally double pattern that is useful in derivations and we can do so in at most two equalization steps, as shown next. Theorem 4 is visualized in Figure 2.

Theorem 4 *Let $\mathcal{S}$ be a set of constant-functional constraints, let $C = (P, L \to R)$ be a functional constraint, let $D = f_1(P) \cup f_2(P)$ be the maximally double pattern of P and L, and let $\mathcal{S} \models C$.*

If $\mathfrak{T} = D$ and an equalization step with $C' \in \mathcal{S}$ is possible, then also a sequence of at most two equalization steps with C' is possible that results in a tableau $\mathfrak{T}'' = D'' = f_1''(P'') \cup f_2''(P'')$, a maximally double pattern of P'' and L'' with $L'' \subseteq \mathcal{V}_{P''}$, such that there is a mapping m with $f_1''(P'') = m(f_1(P))$ and $f_2''(P'') = m(f_2(P))$.

Proof First, we consider all the cases in which $C' = (P', L' \to R')$ is a functional constraint. Let $\$r' \in R'$ be a variable and let e_1 and e_2 be a pair of embeddings of P' into $\mathfrak{T} = D$ with $e_1 =_{L'} e_2$ and $e_1(\$r') \neq e_2(\$r')$, meeting the conditions of line 9 of Algorithm 2. Since an equalization step is performed, we have $e_1(\$r') \notin \mathcal{U}$ or $e_2(\$r') \notin \mathcal{U}$.

We can swap the roles of both f_1, f_2, and e_1, e_2. Hence, without loss of generality, we can assume that $e_1(\$r') = f_1(t)$ with $t \in \mathcal{T}_P$, and, for $\$v \in \mathcal{V}_P$, $e_2(\$r') = f_1(\$v)$ or $e_2(\$r') = f_2(\$v)$. First, we consider the cases in which $e_2(\$r') = f_1(\$v)$.

1. $t \in L \cup \mathcal{U}$ and $\$v \in L$. Since $\$v \in L$, we have $f_1(\$v) = f_2(\$v)$. Let $\phi_{f_1(t) \leftrightarrow f_1(\$v)}$ be the equalization performed by the initial equalization step using C' and embeddings e_1 and e_2, resulting in the tableau $\phi_{f_1(t) \leftrightarrow f_1(\$v)}(f_1(P) \cup f_2(P))$, which is equivalent to the tableau

$$\mathfrak{T}'' = (\phi_{f_1(t) \leftrightarrow f_1(\$v)} \circ f_1(P)) \cup (\phi_{f_1(t) \leftrightarrow f_1(\$v)} \circ f_2(P)).$$

By construction, $\phi_{f_1(t) \leftrightarrow f_1(\$v)} \circ f_1$ and $\phi_{f_1(t) \leftrightarrow f_1(\$v)} \circ f_2$ are embeddings of P into $\mathfrak{T}''$ that agree on $L'' = \phi_{t \leftrightarrow \$v}(L) \cap \mathcal{V} = L \setminus \{\$v\}$ and, for all $t \in \mathcal{T}_P \setminus \{\$v\}$, are equal to f_1 and f_2. Hence, we can put $P'' = \phi_{t \leftrightarrow \$v}(P)$, $f_1'' = f_1|_{\mathcal{T}_{P''}}$, $f_2'' = f_2|_{\mathcal{T}_{P''}}$, and $m = \phi_{f_1(t) \leftrightarrow f_1(\$v)}$.

2. $t \in \mathcal{V}_P \cup \mathcal{U}$ and $\$v \notin L$. Since P, $f_1(P)$, and $f_2(P)$ are isomorphic, and f_1 and f_2 are bijections mapping constants to themselves and variables to variables, the functions $\varepsilon_1 = \Phi_{f_2 \leftarrow f_1} \circ e_1$ and $\varepsilon_2 = \Phi_{f_2 \leftarrow f_1} \circ e_2$ are well-defined and are embeddings of P' into $f_2(P)$ that agree on L'. Hence, by construction, $f_2(t) = \varepsilon_1(\$r') \neq \varepsilon_2(\$r') = f_2(\$v)$. Let $\phi_{f_1(t) \leftrightarrow f_1(\$v)}$ be the equalization performed by the initial equalization step using C' and embeddings e_1 and e_2, resulting in the tableau

$\phi_{f_1(t)\hookleftarrow f_1(\$v)}(f_1(P) \cup f_2(P))$. Since $f_1(\$v) \notin \text{range}(f_2)$, this tableau is equivalent to the tableau

$$\mathfrak{T}' = (\phi_{f_1(t)\hookleftarrow f_1(\$v)} \circ f_1(P)) \cup f_2(P).$$

Hence, $f_2(P)$ is unaffected by the initial equalization step and thus a second equalization step is possible using functional constraint C' and embeddings ε_1 and ε_2.

Performing this second equalization $\phi_{f_2(t)\hookleftarrow f_2(\$v)}$ on $\mathfrak{T}'$ results in the tableau $\phi_{f_2(t)\hookleftarrow f_2(\$v)}(\phi_{f_1(t)\hookleftarrow f_1(\$v)}(f_1(P) \cup f_2(P)))$. Since $f_2(\$v) \notin \text{range}(f_1)$, this tableau is equivalent to the tableau

$$\mathfrak{T}'' = (\phi_{f_1(t)\hookleftarrow f_1(\$v),f_2(t)\hookleftarrow f_2(\$v)} \circ f_1(P)) \cup (\phi_{f_1(t)\hookleftarrow f_1(\$v),f_2(t)\hookleftarrow f_2(\$v)} \circ f_2(P)).$$

By construction, $\phi_{f_1(t)\hookleftarrow f_1(\$v),f_2(t)\hookleftarrow f_2(\$v)} \circ f_1$ and $\phi_{f_1(t)\hookleftarrow f_1(\$v),f_2(t)\hookleftarrow f_2(\$v)} \circ f_2$ are embeddings of P into $\mathfrak{T}''$ that agree on $L'' = \phi_{t\hookleftarrow \$v}(L) \cap \mathcal{V} = L$ and, for all $t \in \mathcal{T}_P \setminus \{\$v\}$, are equal to f_1 and f_2. Hence, we can put $P'' = \phi_{t\hookleftarrow \$v}(P)$, $f_1'' = f_1|_{\mathcal{T}_{P''}}$, $f_2'' = f_2|_{\mathcal{T}_{P''}}$, and $m = \phi_{f_1(t)\hookleftarrow f_1(\$v),f_2(t)\hookleftarrow f_2(\$v)}$.

Next we consider the cases in which $e_2(\$r') = f_2(\$v)$.

3. $e_1(\$r') = f_1(\$v)$. From $e_1(\$r') = f_1(\$v)$, $e_2(\$r') = f_2(\$v)$, and $e_1(\$r') \neq e_2(\$r')$, we conclude that $\$v \notin L$. Let $\phi_{f_1(\$v)\hookleftarrow f_2(\$v)}$ be the equalization performed by the initial equalization step using C' and embeddings e_1 and e_2, resulting in the tableau $\phi_{f_1(\$v)\hookleftarrow f_2(\$v)}(f_1(P) \cup f_2(P))$, which is equivalent to the tableau

$$\mathfrak{T}'' = (\phi_{f_1(\$v)\hookleftarrow f_2(\$v)} \circ f_1(P)) \cup (\phi_{f_1(\$v)\hookleftarrow f_2(\$v)} \circ f_2(P)).$$

By construction, $\phi_{f_1(\$v)\hookleftarrow f_2(\$v)} \circ f_1$ and $\phi_{f_1(\$v)\hookleftarrow f_2(\$v)} \circ f_2$ are embeddings of P into $\mathfrak{T}''$ that agree on $L'' = L \cup \{\$v\}$ and, for all $t \in \mathcal{T}_P \setminus \{\$v\}$, are equal to f_1 and f_2. Hence, we can put $P'' = P$, $f_1'' = \phi_{f_1(\$v)\hookleftarrow f_2(\$v)} \circ f_1$, $f_2'' = \phi_{f_1(\$v)\hookleftarrow f_2(\$v)} \circ f_2$, and $m = \phi_{f_1(\$v)\hookleftarrow f_2(\$v)}$.

4. $e_1(\$r') = f_1(t) \neq f_1(\$v)$. Since P, $f_1(P)$, and $f_2(P)$ are isomorphic and f_1 and f_2 are bijections mapping constants to themselves and variables to variables, the functions $\varepsilon_1 = \Phi_{f_1\hookleftarrow f_2} \circ e_1$ and $\varepsilon_2 = \Phi_{f_1\hookleftarrow f_2} \circ e_2$ are well-defined and are embeddings of P' into $f_1(P)$ with $\varepsilon_1 =_{L'} \varepsilon_2$. Since $e_1(\$r') = f_1(t)$ and $e_2(\$r') = f_2(\$v)$ with $t \neq \$v$, we have $e_1(\$r') = f_1(t) = \varepsilon_1(\$r') \neq \varepsilon_2(\$r') = f_1(\$v)$. Hence, if the equalization step with C', e_1, and e_2 is initially possible, then also an equalization step with C', ε_1, and ε_2 is possible. We choose to perform the equalization step with C', ε_1, and ε_2 instead. Hence, we reduce this case to the cases 1 and 2.

Finally, we consider all the cases in which $C' = (P', E')$ is a constant constraint. Let $(c = \$v') \in E'$ be an equality and let e be an embedding of P' into $\mathfrak{T} = P''$ with $c \neq e(\$v')$, meeting the conditions of line 12 of Algorithm 2. Since an equalization step is performed, we have $e(\$v') \notin \mathcal{U}$. Since we can swap the roles of f_1 and f_2, we can assume, without loss of generality, that $e(\$v') = f_1(\$v)$ with $\$v \in \mathcal{V}_P$. This yields us two cases, namely $\$v \in L$ or $\$v \notin L$.

5. $\$v \in L$. Since $\$v \in L$, we have $f_1(\$v) = f_2(\$v)$. Let $\phi_{c\hookleftarrow f_1(\$v)}$ be the equalization performed by the initial equalization step using C' and embedding e, resulting in the tableau $\phi_{c\hookleftarrow f_1(\$v)}(f_1(P) \cup f_2(P))$, which is equivalent to the tableau

$$\mathfrak{T}'' = (\phi_{c\hookleftarrow f_1(\$v)} \circ f_1(P)) \cup (\phi_{c\hookleftarrow f_1(\$v)} \circ f_2(P)).$$

By construction, the functions $\phi_{c\leftarrow f_1(\$v)} \circ f_1$ and $\phi_{c\leftarrow f_1(\$v)} \circ f_2$ are embeddings of P into $\mathfrak{T}''$ that agree on $L'' = \phi_{c\leftarrow \$v}(L) \cap \mathcal{V} = L \setminus \{\$v\}$ and, for all $t \in \mathcal{T}_P \setminus \{\$v\}$, are equal to f_1 and f_2. Hence, we can put $P'' = \phi_{c\leftarrow \$v}(P)$, $f_1'' = f_1|_{\mathcal{T}_{P''}}$, $f_2'' = f_2|_{\mathcal{T}_{P''}}$, and $m = \phi_{c\leftarrow f_1(\$v)}$.

6. $\$v \notin L$. Since P, $f_1(P)$, and $f_2(P)$ are isomorphic, and f_1 and f_2 are bijections mapping constants to themselves and variables to variables, the function $\varepsilon = \Phi_{f_2\leftarrow f_1} \circ e$ is well-defined and is an embedding of P' into $f_2(P)$ with $\varepsilon(\$v') \neq c$. Hence, by construction, $\varepsilon(\$v') = f_2(\$v)$. Let $\phi_{c\leftarrow f_1(\$v)}$ be the equalization performed by the initial equalization step using C' and embedding e, resulting in the tableau $\phi_{c\leftarrow f_1(\$v)}(f_1(P) \cup f_2(P))$. Since $f_1(\$v) \notin \text{range}(f_2)$, this tableau is equivalent to the tableau

$$\mathfrak{T}' = (\phi_{c\leftarrow f_1(\$v)} \circ f_1(P)) \cup f_2(P).$$

Hence $f_2(P)$ is unaffected by the initial equalization step and thus a second equalization step is possible using constant constraint C' and embedding ε.

Performing this second equalization $\phi_{c\leftarrow f_2(\$v)}$ on $\mathfrak{T}'$ results in the tableau $\phi_{c\leftarrow f_2(\$v)}(\phi_{c\leftarrow f_1(\$v)}(f_1(P) \cup f_2(P)))$. Since $f_2(\$v) \notin \text{range}(f_1)$, this tableau is equivalent to the tableau

$$\mathfrak{T}'' = (\phi_{c\leftarrow f_1(\$v),c\leftarrow f_2(\$v)} \circ f_1(P)) \cup (\phi_{c\leftarrow f_1(\$v),c\leftarrow f_2(\$v)} \circ f_2(P)).$$

By construction, $\phi_{c\leftarrow f_1(\$v),c\leftarrow f_2(\$v)} \circ f_1$ and $\phi_{c\leftarrow f_1(\$v),c\leftarrow f_2(\$v)} \circ f_2$ are embeddings of P into $\mathfrak{T}''$ that agree on $L'' = \phi_{c\leftarrow \$v}(L) \cap \mathcal{V} = L$ and, for all $t \in \mathcal{T}_P \setminus \{\$v\}$, are equal to f_1 and f_2. Hence, we can put $P'' = \phi_{c\leftarrow \$v}(P)$, $f_1'' = f_1|_{\mathcal{T}_{P''}}$, $f_2'' = f_2|_{\mathcal{T}_{P''}}$, and $m = \phi_{c\leftarrow f_1(\$v),c\leftarrow f_2(\$v)}$.

This case analysis completes the proof. □

We refer to the sequence of at most two equalization steps performed in Cases 1, 2, 3, 5, and 6 of Theorem 4 as a *symmetry-preserving step* of type 1, 2, 3, 5, and 6, respectively. The name "symmetry-preserving" reflects that this sequence of equalization steps maintains the symmetry between the parts in the tableau originating from $f_1(P)$ and $f_2(P)$, as illustrated in Figure 2. Notice that we do not consider equalization steps performed by Case 4 of Theorem 4. In this case, we have shown that we can always perform other equalization steps that can be represented by a symmetry-preserving step of type 1 or 2.

Observe that the initial tableau in a chase deciding $\mathcal{S} \models C$, with C a functional constraint, is a maximally double pattern of $\text{PTN}(C)$ and $\text{LHS}(C)$. Hence, if equalization steps are possible, then we can apply Theorem 4 inductively to yield a sequence of symmetry-preserving steps that ends when no further equalization steps are possible. We refer to such chases performing only symmetry-preserving steps as *symmetry-preserving chases*.

Corollary 2 *Let $\mathcal{S}$ be a set of constant-functional constraints, let $C = (P, L \to R)$ be a functional constraint. If $\mathcal{S} \models C$, then there is a symmetry-preserving chase that decides $\mathcal{S} \models C$.*

For simulating the initial symmetry-preserving step in a chase deciding $\mathcal{S} \models C$ with C a functional constraint, we use the following three inference rules.

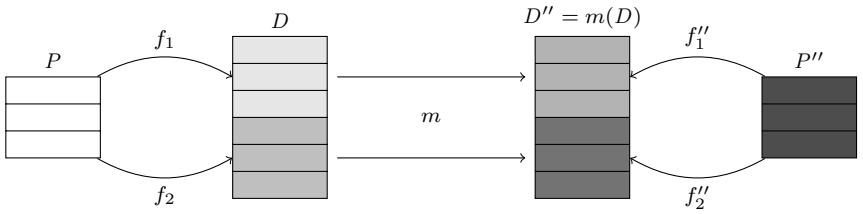

Fig. 2 Visualization of a symmetry-preserving step performed on the maximally double pattern $D = f_1(P) \cup f_2(P)$ of P and L with $L \subseteq \mathcal{V}_P$, resulting in the maximally double pattern $D'' = f_1''(P'') \cup f_2''(P'')$ of P'' and L'' with $L'' \subseteq \mathcal{V}_{P''}$. Since the mapping m is symmetry-preserving, the symmetry between $f_1(P)$ and $f_2(P)$ is maintained: structural changes are applied symmetrically to both halves of D. Hence, m maps each half of D to the corresponding half of D''.

Rule 13 (Application III) *Let P be a pattern, let $\$v \in \mathcal{V}_P$ be a variable, and let $t \in \mathcal{V}_P \cup \mathcal{U}$ be a term. If $(\phi_{t \leftrightarrow \$v}(P), \phi_{t \leftrightarrow \$v}(L) \cap \mathcal{V} \to \phi_{t \leftrightarrow \$v}(R) \cap \mathcal{V})$, if $(P', L' \to R')$, and if there is a pair of embeddings of P' into P that agree on L' and map $\$r' \in R'$ to t and $\$v$, respectively, then $(P, L \to R)$.*

Proof (soundness) Let $\mathcal{R}$ be a relation with $\mathcal{R} \models (\phi_{t \leftrightarrow \$v}(P), \phi_{t \leftrightarrow \$v}(L) \cap \mathcal{V} \to \phi_{t \leftrightarrow \$v}(R) \cap \mathcal{V})$ and $\mathcal{R} \models (P', L' \to R')$, and assume there are embeddings of P into $\mathcal{R}$. Let e_1 and e_2 be any pair of embeddings of P into $\mathcal{R}$ with $e_1 =_L e_2$, and let e_1' and e_2' be embeddings of P' into P that agree on L' and map $\$r' \in R'$ to t and $\$v$, respectively. Now, $g_1 = e_1 \circ e_1'$ and $g_2 = e_1 \circ e_2'$ are embeddings of P' into $\mathcal{R}$ with $g_1 =_{L'} g_2$. By $\mathcal{R} \models (P', L' \to R')$, we have $g_1(\$r') = g_2(\$r')$, and, by construction, we have $g_1(\$r') = e_1(t)$ and $g_2(\$r') = e_1(\$v)$, and, hence, $e_1(\$v) = e_1(t)$. In a similar way, we obtain $e_2(\$v) = e_2(t)$.

As a consequence, $\varepsilon_1 = e_1|_{\text{domain}(e_1) \setminus \{\$v\}}$ and $\varepsilon_2 = e_2|_{\text{domain}(e_2) \setminus \{\$v\}}$ are embeddings of $\phi_{t \leftrightarrow \$v}(P)$ into $\mathcal{R}$. By construction, we have $\varepsilon_1 =_{\phi_{t \leftrightarrow \$v}(L)} \varepsilon_2$ and, hence, $\varepsilon_1 =_{\phi_{t \leftrightarrow \$v}(L) \cap \mathcal{V}} \varepsilon_2$. By $\mathcal{R} \models (\phi_{t \leftrightarrow \$v}(P), \phi_{t \leftrightarrow \$v}(L) \cap \mathcal{V} \to \phi_{t \leftrightarrow \$v}(R) \cap \mathcal{V})$, we conclude $\varepsilon_1 =_{\phi_{t \leftrightarrow \$v}(R) \cap \mathcal{V}} \varepsilon_2$. If $\$v \notin R$, then $R = \phi_{t \leftrightarrow \$v}(R) \cap \mathcal{V}$ and thus $e_1 =_R e_2$. In the other case, when $\$v \in R$, we have $e_1(\$v) = \varepsilon_1(t)$ and $e_2(\$v) = \varepsilon_2(t)$. As t is either a constant or $t \in \phi_{t \leftrightarrow \$v}(R) \cap \mathcal{V}$, we have $\varepsilon_1(t) = \varepsilon_2(t)$. Hence, in both cases we have $e_1 =_R e_2$, and we conclude $\mathcal{R} \models (P, L \to R)$. □

Rule 14 (Application IV) *Let P be a pattern, let $\$v \in \mathcal{V}_P$ be a variable, and let $c \in \mathcal{U}$ be a constant. If $(\phi_{c \leftrightarrow \$v}(P), \phi_{c \leftrightarrow \$v}(L) \cap \mathcal{V} \to \phi_{c \leftrightarrow \$v}(R) \cap \mathcal{V})$, if (P', E') with $(c = \$v') \in E'$, and if there is an embedding of P' into P mapping $\$v'$ to $\$v$, then $(P, L \to R)$.*

Proof (soundness) Let $\mathcal{R}$ be a relation with $\mathcal{R} \models (\phi_{c \leftrightarrow \$v}(P), \phi_{c \leftrightarrow \$v}(L) \cap \mathcal{V} \to \phi_{c \leftrightarrow \$v}(R) \cap \mathcal{V})$ and $\mathcal{R} \models (P', E')$ and assume there are embeddings of P into $\mathcal{R}$. Let e_1 and e_2 be any pair of embeddings of P into $\mathcal{R}$ with $e_1 =_L e_2$, and let e' be an embedding of P' into P mapping $\$v'$ to $\$v$. Now, $e_1'' = e_1 \circ e'$ and $e_2'' = e_2 \circ e'$ are embeddings of P' into $\mathcal{R}$. By $\mathcal{R} \models (P', E')$, we have $c = e_1''(\$v') = e_2''(\$v')$ and, by construction, we have $c = e''(\$v') = e_1(\$v)$ and $c = e_2''(\$v') = e_2(\$v)$, and thus $c = e_1(\$v) = e_2(\$v)$.

As a consequence, $\varepsilon_1 = e_1|_{\text{domain}(e_1) \setminus \{\$v\}}$ and $\varepsilon_2 = e_2|_{\text{domain}(e_2) \setminus \{\$v\}}$ are embeddings of $\phi_{c \leftrightarrow \$v}(P)$ into $\mathcal{R}$. By construction, we have $\varepsilon_1 =_{\phi_{c \leftrightarrow \$v}(L)} \varepsilon_2$ and, hence, $\varepsilon_1 =_{\phi_{c \leftrightarrow \$v}(L) \cap \mathcal{V}} \varepsilon_2$. By $\mathcal{R} \models (\phi_{c \leftrightarrow \$v}(P), \phi_{c \leftrightarrow \$v}(L) \cap \mathcal{V} \to \phi_{c \leftrightarrow \$v}(R) \cap \mathcal{V})$, we

conclude $\varepsilon_1 =_{\phi_{c\hookleftarrow \$v}(R)\cap\mathcal{V}} \varepsilon_2$. If $\$v \notin R$, then $R = \phi_{c\hookleftarrow\$v}(R)\cap\mathcal{V}$ and thus $e_1 =_R e_2$. In the other case, when $\$v \in R$, we already have $e_1(\$v) = e_2(\$v) = c$. Hence, in both cases we have $e_1 =_R e_2$, and we conclude $\mathcal{R} \models (P, L \to R)$. □

The following example exhibits situations in which the Application III and IV rules can be used.

Example 8 Using Application III, we can derive

$$\{(\{(\$a, \$b, \$c)\}, \$a \to \$b), (\{(\$a, b, \$c), (\$a, b, d)\}, \$a \to \$c)\} \vdash \\ (\{(\$a, b, \$c), (\$a, \$b, d)\}, \$a \to \$c)$$

by using the embeddings

$$e_1 = \{\$a \mapsto \$a, \$b \mapsto b, \$c \mapsto \$c\}$$
$$e_2 = \{\$a \mapsto \$a, \$b \mapsto \$b, \$c \mapsto d\}.$$

Using Application IV, we can derive

$$\{(\{(\$a, \$b, \$c)\}, (c = \$c)), (\{(\$a, \$b, c)\}, \$a \to \$b)\} \vdash (\{(\$a, \$b, \$c)\}, \$a \to \$b)$$

by using the embedding $e' = \{\$a \mapsto \$a, \$b \mapsto \$b, \$c \mapsto \$c\}$.

If $(P, (c = \$v))$ is a constant constraint, then, using Application IV, we can also derive $(P, \emptyset \to \$v)$ by using the identity on $\mathcal{V}_P \cup \mathcal{U}$ as the embedding e' of P onto itself.

Rule 15 (Functional Consequence) *Let P be a pattern and let $L \subseteq \mathcal{V}_P$ be a set of variables. If $(P', L' \to R')$ and P' has a pair of embeddings into the maximally double pattern $f_1(P) \cup f_2(P)$ of P and L, agreeing on L' and mapping $\$r' \in R'$ to $f_1(\$v)$ and $f_2(\$v)$, respectively, then $(P, L \to \$v)$.*

Proof (soundness) Let $\mathcal{R}$ be a relation with $\mathcal{R} \models (P', L' \to R')$, and assume there are embeddings of P into $\mathcal{R}$. Let e_1 and e_2 be any pair of embeddings of P into $\mathcal{R}$ with $e_1 =_L e_2$. Let $f_1(P) \cup f_2(P)$ be the maximally double pattern of P and L. Let e'_1 and e'_2 be embeddings of P' into $f_1(P) \cup f_2(P)$ with $e'_1 =_{L'} e'_2$, $e'_1(\$r') = f_1(\$v)$, and $e'_2(\$r') = f_2(\$v)$. As f_1 and f_2 are bijections, the functions $\varepsilon_1 = \Phi_{e_1 \hookleftarrow f_1} \circ \Phi_{e_2 \hookleftarrow f_2} \circ e'_1$ and $\varepsilon_2 = \Phi_{e_1 \hookleftarrow f_1} \circ \Phi_{e_2 \hookleftarrow f_2} \circ e'_2$ are well-defined and are embeddings of P' into $\mathcal{R}$. By construction, we have $\varepsilon_1 =_{L'} \varepsilon_2$, $\varepsilon_1(\$r') = e_1(\$v)$, and $\varepsilon_2(\$r') = e_2(\$v)$. Hence, by $\mathcal{R} \models (P', L' \to R')$, $e_1(\$v) = \varepsilon_1(\$r') = \varepsilon_2(\$r') = e_2(\$v)$. □

The following example exhibits situations in which the Functional Consequence rule can be used.

Example 9 We can derive

$$(\{(\$a, \$b, \$c), (\$x, \$y, \$z)\}, \$b \to \$z) \vdash (\{(\$a, \$b, \$c), (\$x, \$y, \$z)\}, \emptyset \to \$z)$$

using Functional Consequence. We have the maximally double pattern

$$\{(\$a_1, \$b_1, \$c_1), (\$x_1, \$y_1, \$z_1), (\$a_2, \$b_2, \$c_2), (\$x_2, \$y_2, \$z_2)\}$$

of $\{(\$a, \$b, \$c), (\$x, \$y, \$z)\}$ and $\emptyset$. We use the embeddings

$$e_1 = \{\$a \mapsto \$a_1, \$b \mapsto \$b_1, \$c \mapsto \$c_1, \$x \mapsto \$x_1, \$y \mapsto \$y_1, \$z \mapsto \$z_1\}$$
$$e_2 = \{\$a \mapsto \$a_1, \$b \mapsto \$b_1, \$c \mapsto \$c_1, \$x \mapsto \$x_2, \$y \mapsto \$y_2, \$z \mapsto \$z_2\}.$$

We thus have $e_1 =_{\$b} e_2$, $e_1(\$z) = \$z_1$ and $e_2(\$z) = \$z_2$. Hence, using Functional Consequence, we conclude $(\{(\$a, \$b, \$c), (\$x, \$y, \$z)\}, \emptyset \to \$z)$.

Notice that there is a relationship between the Functional Consequence rule and the well-known join dependencies [2] and multivalued dependencies [6]. In this example, the possible embeddings of the pattern $\{(\$a, \$b, \$c), (\$x, \$y, \$z)\}$ can be represented by a relational table T with schema $R(A, B, C, X, Y, Z)$. Due to the construction of T, the join dependency $ABC \bowtie XYZ$ holds on T. This join dependency is equivalent to the multivalued dependency $\emptyset \twoheadrightarrow ABC$. Due to $(\{(\$a, \$b, \$c), (\$x, \$y, \$z)\}, \$b \to \$z)$, the functional dependency $B \to Z$ also holds on T, and, hence, $ABC \to Z$ holds. Using the well-known derivation rules for functional dependencies and multivalued dependencies, we conclude $\emptyset \to Z$.

Before we show how the initial equalization step in any chase deciding $\mathcal{S} \models C$ with C a functional constraint can be simulated by the introduced inference rules, we introduce an additional inference rule to simplify our proofs:

Rule 16 (Embedding II) *If $(P', L' \to R')$ and h is an embedding from P' into P, then $(P, h(L') \cap \mathcal{V} \to h(R') \cap \mathcal{V})$.*

Proof (soundness) Let $\mathcal{R}$ be a relation with $\mathcal{R} \models (P', L' \to R')$, and assume there are embeddings of P into $\mathcal{R}$. Let e_1 and e_2 be any pair of embeddings of P into $\mathcal{R}$ with $e_1 =_{h(L') \cap \mathcal{V}} e_2$. Then, $\varepsilon_1 = e_1 \circ h$ and $\varepsilon_2 = e_2 \circ h$ are embeddings of P' into $\mathcal{R}$. As embeddings always agree on constants and $e_1 =_{h(L') \cap \mathcal{V}} e_2$, we have $e_1 =_{h(L')} e_2$ and, hence, also $\varepsilon_1 =_{L'} \varepsilon_2$. By $\mathcal{R} \models (P', L' \to R')$, we have $\varepsilon_1 =_{R'} \varepsilon_2$. As a consequence, we have $e_1 =_{h(R')} e_2$ and, hence, also $e_1 =_{h(R') \cap \mathcal{V}} e_2$. □

Embedding II explicitly maps a functional constraint defined on a pattern to a different pattern. The following example illustrates this.

Example 10 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 Embedding II with the embedding $h = \{\$a \mapsto \$c, \$b \mapsto \$d\}$.

Next, we show how the initial symmetry-preserving step in any symmetry-preserving chase deciding $\mathcal{S} \models C$ with C a functional constraint can be simulated by the introduced inference rules.

Theorem 5 *Let $\mathcal{S}$ be a set of constant-functional constraints, let $C = (P, L \to R)$ be a functional constraint, and let $\mathcal{S} \models C$.*

If Algorithm 2 is initially able to perform a symmetry-preserving step with $C' \in \mathcal{S}$ resulting in tableau $\mathfrak{T}'' = f_1''(P'') \cup f_2''(P'')$, which is the maximally double pattern of P'' and L'' with $L'' \subseteq \mathcal{V}_{P''}$, then there exists a functional constraint $C'' = (P'', L'' \to R'')$ such that we have the following:

1. *$C \models C''$,*
2. *$\{C', C''\} \vdash C$ using Rules 2, 3, 13, 14, and 15.*

Proof Initially, we have $\mathfrak{T} = f_1(P) \cup f_2(P)$, a maximally double pattern of P and L. By Theorem 4, the initial symmetry-preserving step with constraint C' results in tableau $\mathfrak{T}'' = f_1''(P'') \cup f_2''(P'')$, a maximally double pattern of P'' and L'', such that $f_1''(P'') = m(f_1(P))$ and $f_2''(P'') = m(f_2(P))$ for some mapping m. We make a case distinction on the type of symmetry-preserving step initially performed.

1. The initial symmetry-preserving step is of type 1 or 2. Let $C' = (P', L' \to R')$, let $\$r' \in R'$, and let e_1 and e_2 be the embeddings of P' into $\mathfrak{T}$ with $e_1(\$r') = f_1(t)$ and $e_2(\$r') = f_1(\$v)$, as in the proof of Theorem 4.

We choose $C'' = (\phi_{t \leftarrow \$v}(P), \phi_{t \leftarrow \$v}(L) \cap \mathcal{V} \to \phi_{t \leftarrow \$v}(R) \cap \mathcal{V})$. First, we have $C \models C''$, since we have $C \vdash C''$ by a straightforward application of Embedding II with the embedding $\phi_{t \leftarrow \$v}$. The functions $\varepsilon_1 = {f_1}^{-1} \circ \Phi_{f_1 \leftarrow f_2} \circ e_1$ and $\varepsilon_2 = {f_1}^{-1} \circ \Phi_{f_1 \leftarrow f_2} \circ e_2$ are well-defined and are embeddings of P' into P with, by construction, $\varepsilon_1 =_L \varepsilon_2$, $\varepsilon_1(\$r') = t$, and $\varepsilon_2(\$r') = \$v$. Hence, we can apply Application III using embeddings ε_1 and ε_2, and variable $\$r'$ to derive $\{C', C''\} \vdash C$.

2. The initial symmetry-preserving step is of type 3. Let $C' = (P', L' \to R')$, let $\$r' \in R'$, and let e_1 and e_2 be the embeddings of P' into $\mathfrak{T}$ with $e_1(\$r') = f_1(\$v)$ and $e_2(\$r') = f_2(\$v)$, as in the proof of Theorem 4.

We choose $C'' = (P, L \cup \{\$v\} \to R)$. First, we have $C \models C''$, since we have $C \vdash C''$ by a straightforward application of Augmentation, Decomposition, and Union. A straightforward application of Functional Consequence with e_1, e_2, and $\$v$ yields $C' \vdash (P, L \to \$v)$. Using Augmentation, we obtain $C' \vdash (P, L \to L \cup \{\$v\})$, and, hence, using Transitivity, $\{C', C''\} \vdash C$.

3. The initial symmetry-preserving step is of type 5 or 6. Let $C' = (P', E')$, let $(c = \$v') \in E'$, and let e be the embedding of P' into $\mathfrak{T}$ with $e(\$v') \neq f_1(\$v)$, as in the proof of Theorem 4.

We choose $C'' = (\phi_{c \leftarrow \$v}(P), \phi_{c \leftarrow \$v}(L) \cap \mathcal{V} \to \phi_{c \leftarrow \$v}(R) \cap \mathcal{V})$. First, we have $C \models C''$, since we have $C \vdash C''$ by a straightforward application of Embedding II with the embedding $\phi_{c \leftarrow \$v}$. The function $\varepsilon = {f_1}^{-1} \circ \Phi_{f_1 \leftarrow f_2} \circ e$ is well-defined, and is an embedding of P' into P with, by construction, $\varepsilon(\$v') = \$v$. Hence, we can apply Application IV using embedding ε and variable $\$v'$ to derive $\{C', C''\} \vdash C$.

This case analysis completes the proof. □

5.4 Inference rules for constant-functional constraints

The results from Section 5.1-5.3 only concerned the simulation of direct termination in chases for constant-functional constraints, the initial equalization step in chases for constant constraints, and the initial symmetry-preserving step in chases for functional constraints, respectively. These results are now used as the basis for an induction argument to extend the simulation to the entire chase, as such showing how to translate a chase (constant constraints) or a symmetry-preserving chase (functional constraints) to a derivation.

Proposition 6 *The set of inference rules Reflexivity, Empty, Augmentation, Transitivity, Decomposition, Inconsistency I and II, Application I-IV, Inconsistent Propagation, and Functional Consequence is complete for the class of constant-functional constraints.*

Proof Suppose that $\mathcal{S}$ is a set of constant-functional constraints and C is a single constant-functional constraint with $\mathcal{S} \models C$.

If initially direct termination is possible, then, by Proposition 3-5, Reflexivity, Empty, Inconsistency I, or Inconsistency II can be used to derive C from $\mathcal{S}$.

For chases performing equalization steps we make a case distinction on the type of the constraint C.

1. C is a constant constraint. Let $j > 0$ be the number of equalization steps in a shortest chase deciding $\mathcal{S} \models C$. Assume, as induction hypothesis, that, if $\mathcal{S} \models \overline{C}$, with $\overline{C}$ a constant constraint, and if this can be decided by a chase performing less than j equalization steps, then $\mathcal{S} \vdash \overline{C}$.

Choose a chase deciding $\mathcal{S} \models C$ in exactly j equalization steps. By Theorem 3, there is a constant constraint C'' with $\mathcal{S} \models C''$ and $\{C', C''\} \vdash C$ using Decomposition, Application I, Application II, and Inconsistent Propagation such that the result of the initial equalization step is a tableau $\mathfrak{T}''$, which is equivalent to the initial tableau in any chase deciding $\mathcal{S} \models C''$. Hence, the remaining chase of $j - 1$ equalization steps is a chase deciding $\mathcal{S} \models C''$, and, by the induction hypothesis, $\mathcal{S} \vdash C''$. Since $\mathcal{S} \vdash C''$ and $\{C', C''\} \vdash C$, we conclude $\mathcal{S} \vdash C$.

2. C is a functional constraint. By Corollary 2, we only have to consider symmetry-preserving chases. Let $j > 0$ be the number of symmetry-preserving steps in a shortest symmetry-preserving chase deciding $\mathcal{S} \models C$. Assume, as induction hypothesis, that, if $\mathcal{S} \models \overline{C}$, with $\overline{C}$ a functional constraint, and if this can be decided by a symmetry-preserving chase performing less than j symmetry-preserving steps, then $\mathcal{S} \vdash \overline{C}$.

Choose a symmetry-preserving chase deciding $\mathcal{S} \models C$ in exactly j symmetry-preserving steps. By Theorem 5, there is a functional constraint C'' with $C \models C''$ and $\{C', C''\} \vdash C$ using Augmentation, Transitivity, Application III, Application IV, and Functional Consequence such that the result of the initial symmetry-preserving step is a tableau $\mathfrak{T}''$, which is equivalent to the initial tableau in any chase deciding $\mathcal{S} \models C''$. Hence, the remaining chase of $j - 1$ symmetry-preserving steps is a symmetry-preserving chase deciding $\mathcal{S} \models C''$, and, by the induction hypothesis, $\mathcal{S} \vdash C''$. Since $\mathcal{S} \vdash C''$ and $\{C', C''\} \vdash C$, we conclude $\mathcal{S} \vdash C$.

This case analysis completes the proof. □

Thus, the provided set of inference rules is sound and complete. Finally, we prove that the provided set of inference rules is 2-ary and that each inference rule is indeed decidable. As a consequence, the provided set of inference rules is an axiomatization for the constant-functional constraints.

Theorem 6 *The set of inference rules Reflexivity, Empty, Augmentation, Transitivity, Decomposition, Inconsistency I and II, Application I-IV, Inconsistent Propagation, and Functional Consequence is an axiomatization for the class of constant-functional constraints.*

Proof Each time we introduced an inference rule, we proved its soundness. Proposition 6 proves that the set of inference rules is complete for deriving constant-functional constraints. Hence, it only remains to prove that the inference rules are at most k-ary, for some $k \geq 0$, and that applicability of each inference rule is decidable. The inference rules Reflexivity and Empty are 0-ary. The inference rules

Augmentation, Decomposition, Inconsistent Propagation and Functional Consequence are 1-ary. The inference rules Transitivity, Inconsistency I and II, and Application I-IV are 2-ary.

Applicability of an inference rule can be decided by checking if patterns are equal, which is straightforward, or by checking if embeddings from pattern P into Q exist that satisfy certain properties. Functionally, each embedding from a pattern P into Q can be specified by the mapping from $\mathcal{V}_P$ to $\mathcal{T}_Q$. Since $\mathcal{V}_P$ and $\mathcal{T}_Q$ are finite, we can easily enumerate all possible embeddings and check whether embeddings that satisfy the conditions of an inference rule exist. Hence, checking if an inference rule can be applied is decidable. □

We observe that the Embedding I and II rules are not part of the axiomatization, as they are not used in the simulation of a chase. Their soundness was only used to simplify certain proofs. Likewise, the Union rule is not part of the axiomatization.

By carefully restricting the simulation of chases by inference rules, as presented in Theorem 6, to either chases involving only constant constraints or chases involving only functional constraints, respectively, we can also provide axiomatizations for only the functional constraints and only the constant constraints.

Theorem 7 *The set of inference rules Reflexivity, Augmentation, Transitivity, Inconsistency I*[2]*, Application III, and Functional Consequence is an axiomatization for the class of functional constraints.*

Theorem 8 *The set of inference rules Empty, Decomposition, Inconsistency II*[3]*, Application II and Inconsistent Propagation is an axiomatization for the class of constant constraints.*

Next, we investigate the complexity of the provided axiomatization for the constant-functional constraints.

Theorem 9 *Let $\mathcal{S}$ be a set of constant-functional constraints and let C be a single constant-functional constraint. If $\mathcal{S} \vdash C$, then there is a derivation that performs $\mathcal{O}(|\mathcal{V}_{\mathrm{PTN}(C)}|)$ inference steps.*

Proof Consider the chase deciding $\mathcal{S} \models C$. At each step of the chase, the number of distinct variables in the tableau $\mathfrak{T}$ decreases by one. If C is a constant constraint, then we initially have $|\mathcal{V}_{\mathfrak{T}}| = |\mathcal{V}_P|$. Hence, the chase performs at most $|\mathcal{V}_P|$ equalization steps and Proposition 6 provides a way to simulate each equalization step by a bounded number of inference steps. If C is a functional constraint, then we initially have $|\mathcal{V}_{\mathfrak{T}}| \leq 2|\mathcal{V}_P|$. Hence, the chase performs at most $2|\mathcal{V}_P|$ symmetry-preserving steps and Proposition 6 provides a way to simulate each symmetry-preserving step by a bounded number of inference steps. □

[2] The Inconsistency I inference rule can be used to derive both functional constraints and constant constraints. In this setting we only use Inconsistency I to derive functional constraints.

[3] The Inconsistency II inference rule can be used to derive both constant constraints and functional constraints. In this setting we only use Inconsistency II to derive constant constraints.

6 Related work

Many types of constraints have been investigated for the relational model, and among the simplest of these constraints are the functional dependencies [13]. Functional dependencies play an important role in the well-known Boyce-Codd normal form [14] and in relational schema normalization in general. Besides the functional dependencies, many other types of constraints have been investigated, and for the relational model most of these constraints can be categorized as a subclass of the equality-generating and/or tuple-generating dependencies [1,16,17,26,29]. The constraints studied in this work are all equality-generating. In the following, we shall describe how the constant-functional constraints are related with other classes of equality-generating dependencies.

For describing these relationships, we introduce some terminology to define restrictions on the classes of functional, constant, and equality-generating constraints. We say that a constraint C over pattern P is *typed* if, for every pair of tuples $(t_1, \ldots, t_n) \in P$ and $(u_1, \ldots u_n) \in P$, and for every pair of variables $t_i \in \mathcal{V}_P, u_j \in \mathcal{V}_P$, we have $t_i = u_j$ only if $i = j$. We say that a functional constraint $(P, L \to R)$ is *constant-free* if $\mathcal{U}_P = \emptyset$. We say that an equality-generating constraint (P, E) is *constant-free* if $\mathcal{U}_P = \emptyset$ and, for every $(t = u) \in E$, $t, u \in \mathcal{V}_P$. We say that an equality-generating constraint (P, E) is *many sorted* if it is typed and, for every $(t = u) \in E$, there exist tuples $(t_1, \ldots, t_n) \in P$, $(u_1, \ldots, u_n) \in P$ and there exists a j, $1 \leq j \leq n$, such that $t = t_j$ and $u = u_j$. Lastly, we say that an equality-generating or functional constraint C is an *n-pattern constraint* if $|\textsc{ptn}(C)| \leq n$.

Using the terminology introduced above, we can describe the relationships between various well-known classes of equality-generating constraints. A summary of these relationships is visualized in Figure 3. The *equality-generating constraints* (EGC) and the *functional constraints* (FC) of Akhtar et al. [3] were both originally introduced for the RDF data model, and have been generalized to relations of arbitrary arity in this work. The equality-generating constraints on relations of arbitrary arity are equivalent to the *full equality-generating dependencies* of Wijsen [31]. The *equality-generating dependencies* (EGD) of Beeri and Vardi [7] are equivalent to the constant-free fragment of the equality-generating constraints.

In the literature on dependencies for the relational model, one often considers a n-ary relation to be a subset of $\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$, where $\mathcal{D}_i$, $1 \leq i \leq n$, is a domain of values disjoint from all other domains. As such, many dependencies are many-sorted. The *implication dependencies* of Fagin [16] are an example, and the equality-generating fragment of these implication dependencies are indeed equivalent to the many-sorted and constant-free equality-generating constraints.

Also the well-known *functional dependencies* (FD) of Codd [13] are many-sorted. It is straightforward to verify that the functional dependencies are also equivalent to the typed, constant-free, 1-pattern *functional constraints* (1-pattern FD). Using this relationship, it is also straightforward to verify that the functional dependencies are equivalent to the constant-free, many-sorted, 2-pattern equality-generating dependencies.

In the literature, several context-dependent generalizations of the functional dependencies have been studied. Among them are the *conditional functional dependencies* (CFD) of Fan et al. [21]. These dependencies allow the use of context-dependent information in the specification of integrity constraints, and, as such,

allow for richer tools to maintain integrity of databases. In a straightforward manner, the conditional functional dependencies in *normal form* [21] can be shown to be equivalent to the union of the typed, 1-pattern subset of the *constant constraints* (CD) and the typed, 1-pattern *functional constraints*. Hence, the *constant-functional constraints* are a proper untyped generalization of the conditional functional dependencies towards arbitrary patterns in relations. The *extended conditional functional dependencies* of Bravo et al. [9,20] have been introduced as an extension of the conditional functional dependencies in which one can also express that the valuation of an attribute (in some context) is restricted to a set of allowed values.

Example 11 Consider again the relational schema $PI(name, country, cc, phone)$ of Example 3. The attribute *cc* gives the *country code*, and the set of allowed values can be restricted to all existing country calling codes.

Such an allowed set of values is however not expressible by a set of equalities that should all hold at the same time, and, hence, these types of constraints are not expressible by constant-functional constraints.

Another example of constraints that allow the specification of context-dependent information are the *qualified functional dependencies* (QFD) of He et al. [24]. The qualified functional dependencies introduce a convenient syntax to specify context-dependent functional dependencies on several relations with equivalent schemas in a straightforward manner. It is straightforward to show that each qualified functional dependency holding on each individual relation can be expressed by a set of typed, 1-pattern functional constraints on the relation, and vice versa. Hence, the qualified functional dependencies are also equivalent to the functional part of the conditional 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 [18] and, for the XML data model, DTDs [10] and XSDs [19]. The usage of dependency-like constraints is less common for these data-models although initial steps have been made (e.g., [3,4,11,12,22,23,27,28,30,32]).

We repeat that functional constraints were originally introduced for the RDF data model [3,15]. Besides this obvious relationship with the RDF data model, the concept of applying functional dependencies or functional-like dependencies to a view on the data plays a major role in other proposals for constraints for the RDF and XML data models. A clear and recent example are the *XML Constraints based on XML-to-relational mappings* of Niewerth et al. [28], who studied functional dependencies over tree patterns. Specifically, the tree patterns using only edges are equivalent to patterns, as we consider them, over the edge relation of XML documents.

7 Conclusions and future work

Starting from functional and equality-generating constraints for the RDF data model, we studied constant-functional constraints on arbitrary relations. As our main result, we prove the existence of a sound and complete axiomatization for the constant-functional constraints. This axiomatization can easily be specialized to only deal with functional constraints; hence our results solve a major open problem in the work of Akhtar et al. [3] on functional constraints for the RDF data

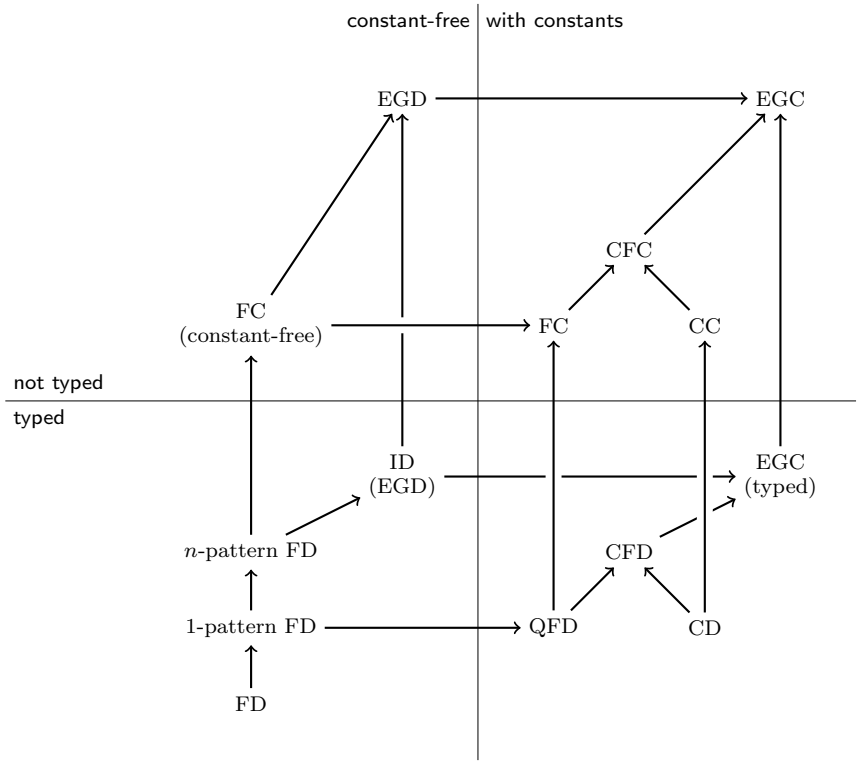

Fig. 3 Overview of the various classes of constraints. Directed edges from a class C_1 to another class C_2 express inclusion of C_1 into C_2, meaning that every constraint in C_1 is expressible by a set of constraints in C_2.

model. The major result leading to the axiomatization of the constant-functional constraints is that chases for functional constraints can always be normalized to symmetry-preserving chases.

We believe that our work provides a promising formal basis for reasoning about constant-functional constraints. As for future work, several open problems remain.

Firstly, for several types of equality-generating constraints, it is known whether Armstrong-like relations [5,16] exist. An Armstrong-like relation $\mathcal{R}$ for a set of constraints $\mathcal{S}$ satisfies constraint C if and only if $\mathcal{S} \models C$. For both the equality-generating constraints and the constant-functional constraints, it is unknown under which conditions Armstrong-like relations exist.

Secondly, the addition of constant constraints to the functional constraints also introduced the possibility to define a set of inconsistent constraints, such that no non-empty relation can satisfy the constraints. An example is the set

$$\{(\{(\$a, \$b, \$c)\}, (c_1 = \$c)), (\{(\$a, \$b, \$c)\}, (c_2 = \$c))\}.$$

Being able to decide consistency of a set of constraints is useful. Inconsistent sets of constraints are not very useful in practice and might even hint at a mistake in the definition of one or more constraints. The complexity to decide consistency of a set of constraints $\mathcal{S}$ is unknown, however.

Thirdly, the exact complexity of the implication problem for constant-functional constraints is not yet known. However, the relationship with conditional functional dependencies already provides lower bounds for the complexity of the consistency problem and the implication problem: for the conditional functional dependencies, these problems are NP-complete and coNP-complete, respectively [21].

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Aho, A.V., Beeri, C., Ullman, J.D.: The theory of joins in relational databases. ACM Transactions on Database Systems **4**(3), 297–314 (1979)
3. Akhtar, W., Cortés-Calabuig, Á., Paredaens, J.: Constraints in RDF. In: Semantics in Data and Knowledge Bases, *Lecture Notes in Computer Science*, vol. 6834, pp. 23–39. Springer (2011)
4. Arenas, M., Libkin, L.: A normal form for XML documents. ACM Transactions on Database Systems **29**(1), 195–232 (2004)
5. Armstrong, W.W.: Dependency structures of data base relationships. In: Information Processing 74, pp. 580–583 (1974)
6. 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, pp. 47–61 (1977)
7. Beeri, C., Vardi, M.: The implication problem for data dependencies. In: Automata, Languages and Programming, *Lecture Notes in Computer Science*, vol. 115, pp. 73–85. Springer (1981)
8. Beeri, C., Vardi, M.Y.: A proof procedure for data dependencies. Journal of the ACM **31**(4), 718–741 (1984)
9. Bravo, L., Fan, W., Geerts, F., Ma, S.: Increasing the expressivity of conditional functional dependencies without extra complexity. In: ICDE '08 Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, pp. 516–525 (2008)
10. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible markup language (XML) 1.0 (fifth edition), W3C recommendation 26 november 2008. URL http://www.w3.org/TR/2008/REC-xml-20081126/
11. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.C.: Keys for XML. Computer Networks **39**(5), 473–487 (2002)
12. Calbimonte, J.P., Porto, F., Keet, C.M.: Functional dependencies in OWL ABOX. In: XXIV Simpósio Brasileiro de Banco de Dados, pp. 16–30 (2009)
13. Codd, E.F.: Relational completeness of data base sublanguages. Tech. Rep. RJ 987, IBM Research Laboratory, San Jose, California (1972)
14. Codd, E.F.: Recent investigations in relational data base systems. In: Information Processing 74, pp. 1017–1021 (1974)
15. Cortés-Calabuig, A., Paredaens, J.: Semantics of constraints in RDFS. In: Proceedings of the 6th Alberto Mendelzon International Workshop on Foundations of Data Management, pp. 75–90 (2012)
16. Fagin, R.: Horn clauses and database dependencies. Journal of the ACM **29**(4), 952–985 (1982)
17. Fagin, R., Vardi, M.Y.: The theory of data dependencies—a survey. In: Mathematics of Information Processing, *Proceedings of Symposia in Applied Mathematics*, vol. 34, pp. 19–71. American Mathematical Society (1986)
18. Fallside, D.C., Walmsley, P.: RDF schema 1.1, W3C recommendation 25 february 2014. URL http://www.w3.org/TR/2014/REC-rdf-schema-20140225/
19. Fallside, D.C., Walmsley, P.: XML schema part 0: Primer second edition, W3C recommendation 28 october 2004. URL http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/
20. Fan, W., Geerts, F., Jia, X.: Conditional dependencies: A principled approach to improving data quality. In: Dataspace: The Final Frontier, *Lecture Notes in Computer Science*, vol. 5588, pp. 8–20. Springer (2009)
21. Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for capturing data inconsistencies. ACM Transactions on Database Systems **33**(2), 6:1–6:48 (2008)

22. Hartmann, S., Link, S.: More functional dependencies for XML. In: Advances in Databases and Information Systems, *Lecture Notes in Computer Science*, vol. 2798, pp. 355–369. Springer (2003)
23. Hartmann, S., Link, S.: Efficient reasoning about a robust XML key fragment. ACM Transactions on Database Systems **34**(2), 10:1–10:33 (2009)
24. 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, pp. 12–21. ACM (2004)
25. Hellings, J., Gyssens, M., Paredaens, J., Wu, Y.: Implication and axiomatization of functional constraints on patterns with an application to the RDF data model. In: Foundations of Information and Knowledge Systems, *Lecture Notes in Computer Science*, vol. 8367, pp. 250–269. Springer (2014)
26. Kanellakis, P.C.: Elements of relational database theory. Tech. Rep. CS-89-39, Brown University (1989)
27. 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, pp. 499–509 (2008)
28. Niewerth, M., Schwentick, T.: Reasoning about XML constraints based on XML-to-relational mappings. In: ICDT, pp. 72–83 (2014)
29. Vardi, M.Y.: Fundamentals of dependency theory. In: Trends in Theoretical Computer Science, *Principles of Computer Science Series*, vol. 34, pp. 171–224. Computer Science Press (1987)
30. 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), 1045–1098 (2012)
31. Wijsen, J.: Database repairing using updates. ACM Transactions on Database Systems **30**(3), 722–768 (2005)
32. Yu, Y., Heflin, J.: Extending functional dependency to detect abnormal data in RDF graphs. In: The Semantic Web ISWC 2011, *Lecture Notes in Computer Science*, vol. 7031, pp. 794–809. Springer (2011)