

On the Decidability and Axiomatization of Query Finiteness in Deductive Databases*

Michael Kifer[†]
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794
kifer@cs.sunysb.edu

April 21, 1998

Abstract

A database query is *finite*¹ if its result consists of a finite set of tuples. For queries formulated as sets of pure Horn rules, the problem of determining finiteness is, in general, undecidable.

In this paper we consider *superfiniteness*—a stronger kind of finiteness, which applies to Horn queries whose function symbols are replaced by the abstraction of *infinite relations* with *finiteness constraints* (abbr., FC’s). We show that superfiniteness is not only decidable but also *axiomatizable*, and the axiomatization yields an effective decision procedure. Although there are finite queries that are not superfinite, we demonstrate that superfinite queries represent an interesting and nontrivial subclass within the class of all finite queries.

Then we turn to the issue of inference of finiteness constraints—an important practical problem that is instrumental in deciding if a query is evaluable by a bottom-up algorithm. Although it is not known whether FC-entailment is decidable for sets of function-free Horn rules, we show that *super-entailment*, a stronger form of entailment, is decidable. We also show how a decision procedure for super-entailment can be used to enhance tests for query finiteness.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Systems – *query processing*; I.2.3 [**Computing Methodologies**]: Deduction and Theorem Proving – *logic programming*.

General Terms: Algorithms, languages, theory.

Additional Keywords and Phrases: Query processing, finiteness constraints, finite queries, horizontal decompositions, partial constraints, computability, axiomatization.

1 Introduction

Query evaluation has been central to deductive database research since the inception of the field. It is known that queries specified via sets of function-free Horn rules are evaluable in finite time. However,

*A preliminary report with some of the results in this paper has appeared in [19].

[†]Work sponsored in part by NSF grants DCR-8603676, IRI-8903507, and CCR-9102159.

¹Finiteness is often called “safety”. We prefer “finiteness” over “safety,” as it is more descriptive and less overloaded.

function-free rules have limited expressive power and advanced deductive database systems, such as LDL [25], Coral [27, 28], and XSB [33], do not prohibit the use of function symbols.

Unfortunately, function symbols make the class of all queries expressible via sets of Horn rules (*Horn queries*) *non-capturable* [18], *i.e.*, there is no single algorithm that can take any set of database facts and a *finite* query—*i.e.*, a query with only a finite number of answers for any finite set of database facts—and will terminate after finding all answers to the query.

An important issue, therefore, is the design of algorithms that can capture various interesting subclasses of finite queries. A special case of this problem is the question of whether a database query is *finite*. Finiteness is known to be undecidable for first-order queries without function symbols (but with negation) [11, 40].² A sound (albeit incomplete) algorithm for testing finiteness for this class of queries was proposed in [41]. Finiteness is also undecidable for Horn queries with function symbols [36]. However, this problem is decidable for Horn queries without function symbols, even in the presence of infinite relations [16].

As finiteness is undecidable in the presence of function symbols, [26] proposed an elegant abstraction of this problem using *function-free* Horn queries with *infinite relations* and *finiteness constraints* (abbr., FC's). It is generally agreed now that this abstraction is a useful tool for studying the problems of finiteness and computability when function symbols must be taken into account. The interest in finiteness constraints was further stimulated by the works [30, 20, 17] that showed that the knowledge of certain FC's over some database predicates may help prove that the query is evaluable by a bottom-up algorithm in a finite number of steps. These studies, thus, suggest that algorithms for FC-inference may become an important part of query processing.

The first step towards the theory of FC-inference was made in [26], where it was noted that finiteness constraints over a single relation have axiomatization similar to Armstrong's axioms for functional dependencies (abbr., FD's) [37]. Nevertheless, it was clear that this result had a long way to go before it could be used for Horn queries, as the latter involve multiple predicates that are intricately connected via logical rules. This paper fills in the missing parts by providing an axiomatization for FC's over multiple predicates and by developing a powerful algorithm for FC-inference. As a special case, these results apply to query finiteness, which we consider first.

Let \mathbf{P} be a set of Horn rules (the *intensional* part of the database) and \mathbf{edb} be a set of database facts (the *extensional* part of the database). Let $T_{\mathbf{P} \cup \mathbf{edb}}$ be the corresponding immediate-consequence operator [22] that maps Herbrand interpretations of \mathbf{P} to other such interpretations. Then any fixpoint of $T_{\mathbf{P} \cup \mathbf{edb}}$ is a model of $\mathbf{P} \cup \mathbf{edb}$ and thus of \mathbf{P} [22]. We call such models *fixpoint models* of \mathbf{P} (note: \mathbf{edb} is a variable parameter here). Fixpoints of $T_{\mathbf{P} \cup \mathbf{edb}}$ are sometimes also called *supported models* [2], and the class of all such models is known to coincide with the class of all models of Clark's completion [22] of $\mathbf{P} \cup \mathbf{edb}$.

For an example of a model that is *not* a fixpoint model, consider the following rule: $p(X) \leftarrow q(X)$. The assignment of the empty relation to q and of the relation $\{< 7 >\}$ to p constitutes a model, but not a fixpoint model. A query defined by a set of Horn rules, \mathbf{P} , is *superfinite* if the query predicate has a finite number of tuples in every fixpoint model of \mathbf{P} . Clearly, a query is finite if it is superfinite.

²These papers dealt with *domain independence*, but a simple modification of the proofs shows undecidability of finiteness as well [16, 17].

Although the converse is not always true, superfinite queries represent an interesting and nontrivial subclass within the class of all finite queries. We show that superfiniteness is not only decidable, but also axiomatizable, and that the axiomatization leads to an effective decision procedure.

An interesting aspect of our axiomatization is that it involves *inclusion dependencies* (IND’s) [5] in addition to finiteness constraints. It follows from our results that despite a number of common properties of FD’s and FC’s, including the common Armstrong’s axiomatization, the inference problem for FC’s and IND’s is decidable *and* axiomatizable. In contrast, the corresponding problem for FD’s and IND’s is neither axiomatizable (in the conventional sense) [5], nor decidable [24, 6]. This may seem even more surprising in view of the fact that FC’s are not first-order entities (*i.e.*, they cannot be expressed as a first-order formula), while the notion of an FD is first-order.

Applying these ideas to FC-inference, we define the notion of *super-entailment*, where an FC α is super-entailed by a database \mathbf{P} and by a set F of FC’s over the predicates of \mathbf{P} , if α holds in every fixpoint of \mathbf{P} that satisfies every constraint in F . We show that, like superfiniteness, super-entailment is decidable and axiomatizable.

In [26], it was claimed that finiteness is decidable for Horn queries with FC’s and infinite relations. Unfortunately, it was later discovered that the proposed algorithm is incomplete. In [32] the finiteness problem was split into *weak finiteness*³ and *termination*. It was then shown that weak finiteness—a property of Horn queries that guarantees that all intermediate relations remain finite in the course of a bottom-up fixpoint computation—is decidable. It was also shown that finiteness is decidable in polynomial time for monadic Horn queries that admit infinite relations and FC’s. However, it is still unknown whether termination is decidable for weakly finite databases, so the problem of finiteness in the presence of FC’s still eludes solution. It is also not known whether logical entailment of FC’s (as opposed to super-entailment) is decidable. We note that finiteness and FC-inference are closely related: finiteness can be specified as a special kind of FC (Section 4), so the finiteness problem is a special case of FC-inference.

This paper is organized as follows. Section 2 introduces notation and terminology. In Sections 3 through 8 we motivate and develop the machinery needed to prove the axiomatizability and decidability of superfiniteness and super-entailment. The main results are proved in Sections 9 and 10. Section 11 surveys state of the art with respect to the finiteness problem and lists some open problems. Section 12 shows how the results of this paper can be combined with other methods to yield more powerful tests for query finiteness. Section 13 concludes the paper.

2 Preliminaries

For future reference, we shall briefly review the standard notions of logic programming and deductive databases. The reader is referred to [22, 38, 39] for more details.

³In [32] this property is called “weak safety.”

Terms, Rules, Databases

A *term* is either a variable, or a function symbol applied to the number of terms appropriate for the arity of that symbol. For instance, if f is a function symbol of arity k and t_1, \dots, t_k are terms then $f(t_1, \dots, t_k)$ is a term. A *constant* is a 0-ary function symbol. If c is a constant, it is customary to identify c with the term $c()$. For the most part, the only terms considered in this paper will be either constants or variables. So, when we refer to function symbols rather than just constants, we shall mean function symbols of arity 1 or higher.

Expressions of the form $p(t_1, \dots, t_n)$, where p is a predicate symbol of arity n and t_1, \dots, t_n are terms, is called an *atomic* formula (or, simply, an *atom*). A *Horn rule* has the form

$$p(t) \leftarrow q_1(t_1), q_2(t_2), \dots, q_n(t_n) \quad (1)$$

where $p(t)$ and the $q_i(t_i)$'s are atoms. Here and henceforth we shall use the (possibly subscripted) symbols $p, q, r; V, W, X, Y, Z;$ and t, s to denote predicate symbols, variables, and terms, respectively. The atom $p(t)$ is called the *head* of the rule and the rest of the atoms in (1) are called *body literals*.

An atom, a term, or a rule is *ground* if it has no variables. Atoms can also be viewed as empty-bodied rules. Ground atoms are called *facts*. A *ground instance* of the rule (1) above is any rule obtained from (1) by a *consistent*, simultaneous substitution of variables with ground terms. A substitution is consistent if different occurrences of the same variable are replaced with the same term (and it is simultaneous if all replacements are performed simultaneously, not one after another).

A *database* is any set of Horn rules. The set of all facts in a database is called the *extensional database* (abbr., *EDB*); the set of remaining rules is the *intensional database* (abbr., *IDB*). We assume that the only empty-bodied rules in the database are ground facts. It follows, then, that all IDB-rules have non-empty body.

Henceforth, we reserve the term “rule” for rules with non-empty bodies. The IDB part of the database will be usually denoted by \mathbf{P} , while the EDB part will be denoted by \mathbf{edb} . The symbol \mathcal{D} will be used for the union of the two.

Without loss of generality, we assume that EDB and IDB have disjoint sets of predicates. We will therefore be justified in calling the predicate symbols in the EDB, the *EDB-predicates*; predicate symbols that occur in the IDB only will be called *IDB-predicates*. EDB-predicates may have an infinite number of facts. These *infinite predicates* are used to represent arithmetic operations and to simulate function symbols.

A set of *integrity constraints* (abbr., *IC*) may be specified over the EDB-predicates, in which case EDB-predicates must satisfy these constraints. For instance, an integrity constraint may be that the extension of a certain predicate must be finite in every EDB. The set *IC* together with the information about the predicates used in \mathcal{D} (*i.e.*, predicate names and their arities) comprise the *schema* of the database \mathcal{D} .

Models and Fixpoints

We assume some fixed alphabet \mathcal{L} of function and predicate symbols. A *Herbrand universe* is the set of all ground terms constructible in \mathcal{L} . In this paper, we assume an arbitrary but fixed Herbrand

universe. For the most part, our databases will have no function symbols of positive arity, but we shall assume that \mathcal{L} contains an infinite number of constants. Thus, the Herbrand universe will still be infinite. However, for simplicity, the number of predicate symbols in \mathcal{L} is assumed to be finite.

A Herbrand *base* is the set of *all* ground atoms constructible in \mathcal{L} . A Herbrand *interpretation* is any subset of the Herbrand base. Since we do not consider other kinds of universes and interpretations, we shall be using the terms “universe” and “interpretation” exclusively for Herbrand universes and interpretations.

A ground atom is *true* in (or is *satisfied* by) an interpretation I if it belongs to I . A ground rule of the form (1) above is *true* in I if and only if $p(t)$ is true whenever all the $q_i(t_i)$'s (which are ground atoms) are true in I . A non-ground rule is true in I if and only if all of its ground instances (with respect to the given Herbrand universe) are true in I . A *model* of a database \mathcal{D} is any interpretation that satisfies (makes true) every IDB-rule and every EDB-fact in \mathcal{D} .

A well-known fact about Horn databases is that intersection of any number of models of such a database \mathcal{D} is also a model. Intersection of all models is thus the *least* (by inclusion) model of \mathcal{D} [22]. This model can also be characterized as the unique least fixpoint of the operator $T_{\mathcal{D}}$ that maps the interpretations of \mathcal{D} into other such interpretations, as defined next. Let I be an interpretation. We define $T_{\mathcal{D}}$ as follows:

$$T_{\mathcal{D}}(I) \stackrel{\text{def}}{=} \{ p \mid p \text{ is an EDB-fact in } \mathcal{D}; \text{ or there is a ground instance } p \leftarrow q_1, \dots, q_n \text{ of an IDB-rule in } \mathcal{D}, \text{ such that } q_i \in I \text{ for } i = 1, \dots, n \}$$

A *fixpoint* of $T_{\mathcal{D}}$ is any interpretation I such that $T_{\mathcal{D}}(I) = I$. A *pre-fixpoint* is an interpretation such that $T_{\mathcal{D}}(I) \subseteq I$. It is well-known [22] that I is a model of \mathcal{D} if and only if it is a pre-fixpoint of $T_{\mathcal{D}}$; it is the least model of \mathcal{D} if and only if it is the least fixpoint of $T_{\mathcal{D}}$.

If \mathbf{P} is an IDB and \mathbf{edb} is an EDB, any fixpoint of $T_{\mathbf{P} \cup \mathbf{edb}}$ is called a *fixpoint model* of \mathbf{P} (that arises from \mathbf{edb}). Likewise, any least fixpoint of $T_{\mathbf{P} \cup \mathbf{edb}}$ is said to be a *least fixpoint model* of \mathbf{P} (which is also the least model of $\mathbf{P} \cup \mathbf{edb}$ [22]). Note that here \mathbf{edb} acts as a parameter that gives rise to different fixpoint (and least fixpoint) models of \mathbf{P} .

Queries and Answers

A *query* to a database \mathcal{D} is a statement of the form $?- q(X_1, \dots, X_n)$, where q is an n -ary predicate symbol from IDB or EDB, and X_1, \dots, X_n are *distinct* variables. In particular, our queries do not contain constants and function symbols. Although this notion at first seems less general than the usual definition of queries [38], the two are known to be essentially equivalent.

The set of *answers* to the above query is the set of all facts for the query predicate in the least model of \mathcal{D} . More formally, the answer to $?- q(X_1, \dots, X_n)$ is the following set of ground atoms:

$$\{ q(t_1, \dots, t_n) \mid q(t_1, \dots, t_n) \text{ is true in the least model of } \mathcal{D} \}$$

Naming Conventions

It is common to refer to Horn databases that use neither function symbols (other than constants) nor infinite EDB-relations as *Datalog* databases. Augmented with infinite EDB-relations, such databases

are called *Extended Datalog*. In comparative studies of various kinds of databases, it has also become common to talk about “Datalog with function symbols,” although this is somewhat of a misnomer, since the very term “Datalog” was introduced in [23] specifically to indicate the absence of function symbols. The reader is referred to [38, 39] for the introductory material on deductive databases.

Most of this paper is concerned with Extended Datalog. Extended Datalog was introduced in [26] as an approximation of Datalog with function symbols⁴ and has been extensively studied since then [32, 16, 4, 31, 30]. The interest in this approximation is fueled primarily by the fact that finiteness and computability of Extended Datalog queries is easier to study, and the corresponding algorithms can be converted into sufficient tests for finiteness and termination of queries over Horn databases with function symbols.

We use the convention that infinite EDB-predicates are denoted by (possibly subscripted) letters f, g, h ; finite EDB-predicates are denoted by a, b, c, d ; and derived predicates by p, q, r . For variables and relational attributes we shall use capital letters.

We shall use the following terminological conventions. First, the terms *relation name* (or *attribute*) and *predicate symbol* (resp., *argument*) will be used interchangeably. Second, we distinguish between a predicate symbol p and the *relation instance* or *extension* assigned to p by an interpretation. Relation instances will usually be marked with the “bar”, e.g., \bar{p} . Here p is just a symbol in a language, while \bar{p} is the set of facts known to be true of p in some interpretation (which is known from the context). We will also freely alternate between the terms “interpretation” and “database instance”—both will stand for nothing more than an assignment of relations to predicate symbols. When the concrete interpretation is immaterial or is clear from the context, we may talk about relation instances without referring to any specific interpretation.

Let \vec{X} be a sequence of attributes (where the same attribute may have multiple occurrences) of a predicate symbol p and let \bar{p} be a relation instance for p . We shall use $\bar{p}[\vec{X}]$ to denote the *projection* of \bar{p} on \vec{X} (see [37]).

3 Finiteness and Superfiniteness

Informally, a given query is *finite* if it has a finite set of answers for all instances of the EDB that satisfy all integrity constraints supplied with the database. Thus, a query is *infinite* if there is some instance of the EDB that satisfies the integrity constraints for which the query has an infinite set of answers.

More precisely, let \mathbf{P} be an IDB. A query defined by \mathbf{P} is finite with respect to a set of integrity constraints IC if the query has a finite number of answers with respect to the database $\mathbf{P} \cup \mathbf{edb}$ for every \mathbf{edb} that satisfies all the constraints in IC . Alternatively, we can say that finite queries are precisely those whose predicate symbol has a finite extension in every least fixpoint model of \mathbf{P} that satisfies IC . Although, strictly speaking, IC is distinct from \mathbf{P} , it is sometimes convenient to lump the IDB and the integrity constraints together and use \mathbf{P} to denote the combined entity.

⁴For instance, a function-free approximation of $p(f(X)) \leftarrow q(X)$ is $p(Y) \leftarrow q(X), f(X, Y)$, where f is a predicate that admits infinite relations.

Instead of considering extensions of the query predicate with respect to the least fixpoint models, one can consider extensions relatively to *any* fixpoint model of \mathbf{P} . This leads to a stronger notion of finiteness: A query defined by \mathbf{P} is *superfinite* if for every fixpoint model I of \mathbf{P} that arises from an EDB that satisfies all the integrity constraints in IC , the extension of the query predicate is finite in I .⁵

Clearly, superfiniteness implies finiteness, but not vice versa. For instance, if \mathbf{P} is $p(X) \leftarrow p(X)$ then the query $?-p(X)$ is finite, as any least model always assigns the empty extension to p . However, this query is not superfinite, since there are fixpoints of $T_{\mathbf{P} \cup \mathbf{edb}}$ (for any \mathbf{edb}) that assign arbitrary infinite relations to p . Nevertheless, as we shall see, superfinite queries form a useful, non-trivial class.

4 Finiteness Constraints

The main type of integrity constraints to be considered in this paper is the *finiteness constraint*. A *finiteness constraint* (abbr., FC) [26] over an EDB-predicate p is a statement of the form $p : X \dashrightarrow Y$, where X and Y are sets of attributes. The name of the predicate may be omitted if it is immaterial or clear from the context.

Occasionally, we shall denote attributes by their position number in the predicate. For instance, if p is a predicate symbol with the first attribute X and the second attribute Y , we may write $p : 2 \dashrightarrow 1$ in lieu of $p : Y \dashrightarrow X$. This will be called *position-number* notation (as opposed to *attribute-name* notation, in which attributes are referred to by their names). In an FC, the predicate name is explicit and so argument positions can be simply referred to their position number. In other contexts, the name of a predicate may not be explicit, but the position-number notation may still be convenient. When ambiguity may arise, we shall prepend the predicate name to the argument position. For instance, $p : 2$ will denote the second attribute of p .

A relation instance \bar{p} over a predicate p satisfies the FC $p : X \dashrightarrow Y$ if and only if the following property holds: For each tuple $t \in \bar{p}[X]$, the set of tuples in $\sigma_{X=t}(\bar{p})[Y]$ is finite, where $\sigma_{X=t}(\bar{p})$ is the set of all tuples in \bar{p} that have the value t on X . Note that the notion of an FC is strictly weaker than the traditional notion of a functional dependency (abbr., FD)—FCs hold trivially in finite relations.

Example 4.1 (Finiteness constraints) Consider a predicate $f(X, Y)$ over the domain of integers such that $\bar{f} = \{ \langle n, m \rangle \mid n > m > 0 \}$. Then the finiteness constraint $f : 1 \dashrightarrow 2$ holds in \bar{f} , while $f : 2 \dashrightarrow 1$ does not. Indeed, for each n there are only finitely many positive integers smaller than n ; on the other hand, for any m there are infinitely many integers greater than m . \square

As a special case, an FC of the form $p : \{ \} \dashrightarrow X$ holds in a relation \bar{p} if and only if the projection of \bar{p} on X is finite. One use of this special form is to distinguish between finite and infinite EDB-predicates and their arguments. The other use is for proving finiteness of IDB-predicates. Because of our frequent use of finiteness constraints of the form $p : \{ \} \dashrightarrow X$, we shall give them a more

⁵One may wonder if *hyperfiniteness*—a notion akin to superfiniteness, except that the query predicate is required to be finite in *all* models that satisfy the IC's—is of any interest. However, it turns out that no query is hyperfinite in this sense: for any predicate p , a set F of FC's, and a database \mathbf{D} with EDB satisfying F , one can always construct a model of \mathbf{D} where p has an infinite extension.

suggestive notation, $p : \text{Finite}(X)$. This latter type of constraints obviates the need for distinguishing between finite and infinite EDB-predicates. For instance, when we say that $p(X, Y)$ is a finite relation, we simply mean that there is a constraint $p : \text{Finite}(X, Y)$.

Apart from specifying finiteness, FC's can be used to approximate function symbols, which is the main reason behind their introduction in [26]. The idea is to replace each occurrence of a function term $f(X_1, \dots, X_n)$ with a *new* variable X and to add a predicate of the form $f(X_1, \dots, X_n, X)$ to the body of the corresponding rule along with the FC $f : 1, \dots, n \rightarrow (n + 1)$. For instance, $p(f(X, Y)) \leftarrow q(X, g(Z))$ would be converted into

$$p(W) \leftarrow q(X, V), f(X, Y, W), g(Z, V)$$

with the FC's $g : Z \rightarrow V$ and $f : X, Y \rightarrow W$. Details of this process can be found in [26], where it is also shown that finiteness of the transformed query implies finiteness of the original query.

The following is a complete axiomatization for finiteness constraints over a *single* predicate:

FC-Rules:

- (i) **from** $X \subseteq Y$ **infer** $Y \rightarrow X$
- (ii) **from** $X \rightarrow Y, Y \rightarrow Z$ **infer** $X \rightarrow Z$
- (iii) **from** $X \rightarrow Y$ **infer** $(X \cup Z) \rightarrow (Y \cup Z)$, for any set of attributes Z

A brief look at the FC-rules reveals that they are identical to Armstrong's axioms for functional dependencies. The following useful inference rules follow easily from the above:

Auxiliary Rules for Finite(X):

- (i) **from** $\text{Finite}(X), Y \subseteq X$ **infer** $\text{Finite}(Y)$
- (ii) **from** $\text{Finite}(X), X \rightarrow Y$ **infer** $\text{Finite}(Y)$
- (iii) **from** $\text{Finite}(X), \text{Finite}(Y)$ **infer** $\text{Finite}(XY)$
- (iv) **from** $\text{Finite}(Y)$ **infer** $X \rightarrow Y$, for any set of attributes X

If a set of constraints G can be derived from another set of constraints F using the above rules then we write $F \vdash G$. We say that F (semantically) *implies* G , denoted $F \models G$, if G holds in every relation in which F holds. The following result is from [26]; its proof is presented here for completeness of the exposition.

Proposition 4.2 *FC-rules are sound and complete for inferring FC's, i.e., $F \vdash G$ if and only if $F \models G$, for any pair of sets of FC's.*

Proof: The proof follows the lines of the corresponding proof for functional dependencies in [37]. Soundness of the rules is an easy consequence of the definitions. For completeness, let us define the *closure* of X , denoted X_F^+ , as $\{A \mid X \rightarrow A \text{ follows from } F \text{ by the inference rules}\}$.

First, we claim that $X \rightarrow Y$ follows by the FC-rules if and only if Y is a subset of X_F^+ . The proof is identical to the proof of the corresponding fact for functional dependencies [37].

X_F^+	$U - X_F^+$
a a a a	b₁ b₁ b₁ b₁
a a a a	b₂ b₂ b₂ b₂
a a a a	b₃ b₃ b₃ b₃
⋮ ⋮	⋮ ⋮ ⋮

Figure 1: Relation \bar{r}_X

Next, let F be a set of FC's over a set U of attributes and suppose that $X \dashrightarrow Y$ cannot be inferred using the FC-rules. We exhibit a relation \bar{r}_X that satisfies all constraints in F but violates $X \dashrightarrow Y$. Let $\bar{r}_X[X_F^+]$ consist of exactly one tuple, and let $\bar{r}_X[U - X_F^+]$ be infinite. Furthermore, for each tuple of \bar{r}_X , the attributes in $U - X_F^+$ have the same value, which is specific to this tuple and does not occur elsewhere in \bar{r}_X . Relation \bar{r}_X is depicted in Figure 1.

Relation \bar{r}_X satisfies F . Indeed, assume that $V \dashrightarrow W$ is in F , but is violated by \bar{r}_X . By the construction of \bar{r}_X , we have $V \not\subseteq X_F^+$, or else $W \subseteq X_F^+$ and $V \dashrightarrow W$ would be satisfied by \bar{r}_X . But then $V \cap (U - X_F^+) \neq \{\}$. Hence, by the construction of \bar{r}_X , corresponding to each tuple in $\bar{r}_X[V]$ there is exactly one tuple in $\bar{r}_X[W]$. Thus, $V \dashrightarrow W$ holds, contrary to our assumption.

On the other hand, since $X \dashrightarrow Y$ does not follow from F , we have $Y \cap (U - X_F^+) \neq \{\}$. Thus, $X \dashrightarrow Y$ is violated by \bar{r}_X . \square

Corollary 4.3 *Implication and equivalence for sets of FC's is decidable.*

Proof: Let $F \models G$. Then, by Proposition 4.2, $F \vdash G$. It is enough to check that $F \vdash \delta$ for every $\delta \in G$. By the proof of Proposition 4.2, if δ is $X \dashrightarrow Y$ then $F \vdash \delta$ if and only if $Y \subseteq X_F^+$. Closure of any set can be computed in linear time, as in [3], so checking the implication $F \models G$ takes quadratic time in the size of F and G . \square

5 Motivating Examples

In this section, we motivate the need for various integrity constraints and illustrate how they can be used for proving finiteness.

From now on, we shall work exclusively with what we earlier called Extended Datalog, a strain of Datalog that allows infinite relations constrained by sets of FC's. Furthermore, we assume that all our rules are *range-restricted*. This means that every variable that occurs in the head of a rule must also appear in the body of that rule. This requirement is easy to meet by transforming the rules as follows: if X is a non-range-restricted variable in the head of a rule then we add the atom $f(X)$ to the body of the rule, where f is a new predicate that admits infinite extensions.

Example 5.1 (Detecting finiteness using FC's) Let the IDB \mathbf{P} consist of the following rules:

$$\begin{aligned}
R_1 &: r(X) \leftarrow f(X, Y), r(Y), a(Y) \\
R_2 &: r(Z) \leftarrow b(Z)
\end{aligned}$$

Let the FC $f : 2 \dashrightarrow 1$ hold. By our naming convention, a and b denote finite EDB-predicates and f denotes an infinite EDB-predicate (it could have been a result of the elimination of the function symbol from the rule $r(f(Y)) \leftarrow r(Y), a(Y)$).

The finite predicate a in R_1 ensures that no matter how often the rule is applied, the set of all possible values for Y is finite. Since Y finitely determines X (due to the FC $f : 2 \dashrightarrow 1$), the set of all possible values for X is finite. Thus, the query $?- p(X)$ is finite. \square

Let us see how the inference rules for FC's can help in proving query finiteness. In the above example, let M be a fixpoint model of \mathbf{P} . This model associates a binary relation \overline{R}_1 over the attributes X, Y with the rule R_1 , and a unary relation \overline{R}_2 over the single attribute Z with the rule R_2 . These relations are defined as follows:

$$\begin{aligned}
\overline{R}_1(X, Y) &\stackrel{\text{def}}{=} \{ \langle x, y \rangle \mid f(x, y) \in M, r(y) \in M, a(y) \in M \} \\
\overline{R}_2(Z) &\stackrel{\text{def}}{=} \{ \langle z \rangle \mid b(z) \in M \}
\end{aligned}$$

Since a and b are finite database predicates, and because of the FC $f : 2 \dashrightarrow 1$, we have the following constraints on the above relations:

$$\begin{aligned}
\overline{R}_1 &: \text{Finite}(Y), Y \dashrightarrow X \\
\overline{R}_2 &: \text{Finite}(Z)
\end{aligned}$$

By the FC-inference rules (and their derivatives), we conclude that $\text{Finite}(X)$ holds in $\overline{R}_1(X, Y)$. Thus, $\overline{r} = \overline{R}_1[X] \cup \overline{R}_2[Z]$ is finite because each of its components is provably finite, where \overline{r} is the relation that M assigns to r .

The above argument is not completely satisfactory, though, since it is not fully proof-theoretic. The problem is the lack of rules for inferring constraints in a union of two relations from constraints on the components of that union. For instance, \overline{r} is a union of its components, $\overline{R}_1[X]$ and $\overline{R}_1[Z]$, and we reached the conclusion that \overline{r} is finite using the fact that both of its components are provably finite, not via the inference system. This example suggests that we need additional rules of inference to deal with decompositions of relations and with inclusion dependencies between them.

The following non-trivial example illustrates the use of decompositions and inclusion dependencies for finiteness analysis.

Example 5.2 (Finiteness and horizontal decompositions) Let \mathbf{P} be the following IDB:⁶

$$\begin{aligned}
R_1 &: p(X_1, X_1) \leftarrow b(X_1) \\
R_2 &: p(X_2, Y_2) \leftarrow b(Y_2), g(X_2, V_2), h(X_2, W_2), p(V_2, W_2) \\
R_3 &: p(X_3, Y_3) \leftarrow b(X_3), g(Y_3, V_3), h(Y_3, W_3), p(V_3, W_3) \\
g &: 2 \dashrightarrow 1 \\
h &: 2 \dashrightarrow 1
\end{aligned}$$

⁶As an amusing aside, [21] points out that this database has a “real-life” interpretation as a definition of the concept of “founding fathers and mothers.”

where b is a finite EDB-predicate and g, h are infinite EDB-predicates. Is p finite? We begin to argue as in Example 5.1. First, given a fixpoint model M , we construct relations corresponding to the rules (where the names of attributes are listed following the relation names):

$$\begin{aligned}\overline{R}_1(X_1) &\stackrel{\text{def}}{=} \{ \langle x \rangle \mid b(x) \in M \} \\ \overline{R}_2(X_2, Y_2, V_2, W_2) &\stackrel{\text{def}}{=} \{ \langle x, y, v, w \rangle \mid b(y), g(x, v), h(x, w), p(v, w) \in M \} \\ \overline{R}_3(X_3, Y_3, V_3, W_3) &\stackrel{\text{def}}{=} \{ \langle x, y, v, w \rangle \mid b(x), g(y, v), h(y, w), p(v, w) \in M \}\end{aligned}$$

Since b is finite and because of the FC's on g and h , we have the following constraints:

$$\begin{aligned}\overline{R}_1: & \text{Finite}(X_1) \\ \overline{R}_2: & \text{Finite}(Y_2), V_2 \dashrightarrow X_2, W_2 \dashrightarrow X_2 \\ \overline{R}_3: & \text{Finite}(X_3), V_3 \dashrightarrow Y_3, W_3 \dashrightarrow Y_3\end{aligned}$$

Furthermore, we know that \overline{p} , the relation instance assigned to p by M , is composed of three relations that are projections of $\overline{R}_1, \overline{R}_2$, and \overline{R}_3 :

$$\overline{p} = \overline{R}_1[X_1, X_1] \cup \overline{R}_2[X_2, Y_2] \cup \overline{R}_3[X_3, Y_3]$$

The inclusion constraints between p and each of its three components are important for the finiteness analysis. However, to prove that p is finite in M —and finite it is—we need to take a closer look at the components of \overline{p} and at the constraints over them.

From the original constraints, we can conclude that the first component of \overline{p} , $\overline{R}_1[X_1, X_1]$, is finite. The second component, $\overline{R}_2[X_2, Y_2]$, consists of tuples derived by means of rule R_2 . From the constraints we know that these tuples have only a finite number of values in their second attribute Y_2 , but possibly an infinite number of values in the attribute X_2 . The last component of \overline{p} is $\overline{R}_3[X_3, Y_3]$. This relation has only a finite number of values in X_3 , but possibly an infinite number of values in Y_3 .

Since the first component of \overline{p} is finite, showing that the other two are also finite would entail finiteness of \overline{p} . Consider $\overline{R}_2[X_2, Y_2]$. We need to show only that the number of values in the first attribute is finite. To establish this, we examine the genealogy of tuples in $\overline{R}_2[X_2, Y_2]$. These tuples are generated by rule R_2 , and the p -tuples used in the *body* of R_2 must come from one of the three components of \overline{p} . These components of \overline{p} split \overline{R}_2 into three subrelations. The following case-analysis discusses each of these subcomponents separately.

Component 1: Finiteness of $\overline{R}_1[X_1, X_1]$ has already been established. From the FC's $V_2 \dashrightarrow X_2$ and $W_2 \dashrightarrow Y_2$, it is clear that only a finite number of tuples can be generated via rule R_2 , when p is instantiated with $\overline{R}_1[X_1, X_1]$ in the body of R_2 .

Hence, the subcomponent of \overline{R}_2 that comes from the instantiation of \overline{p} with $\overline{R}_1[X_1, X_1]$ in the body of R_2 has a finite projection on the attributes X_2, Y_2 .

Component 2: $\overline{R}_2[X_2, Y_2]$ has not yet been shown finite, but $\overline{R}_2[Y_2]$ is finite. Using the FC $W_2 \dashrightarrow X_2$, we can infer that there is only a finite number of different X_2 -values in the tuples generated by R_2 , when $\overline{R}_2[X_2, Y_2]$ instantiates p in the body.

Hence, the subcomponent of \overline{R}_2 that comes from the instantiation of \overline{p} with $\overline{R}_2[X_2, Y_2]$ in R_2 has a finite projection on X_2, Y_2 .

Component 3: Finiteness of $\overline{R}_3[X_3, Y_3]$ has not yet been established, but $\overline{R}_3[X_3]$ is finite. Using the FC $V_2 \rightarrow X_2$, we can infer that the number of different values in the first attribute of the relation generated by R_2 , when $\overline{R}_3[X_3, Y_3]$ is used in the body for p , is finite.

Thus, the subcomponent of \overline{R}_2 that comes from the instantiation of \overline{p} with $\overline{R}_3[X_3, Y_3]$ in the body of R_2 has a finite projection on X_2, Y_2 .

Similar analysis shows that $\overline{R}_3[X_3, Y_3]$ is finite. Thus, \overline{p} is finite. Since M is an arbitrary fixpoint model of \mathbf{P} , we conclude that p is, in fact, superfinite. \square

In addition to illustrating the importance of decompositions for the finiteness analysis, the above example brings out an important point: When analyzing recursive predicates, we must consider their components separately and, in principle, each component may need to be decomposed even further in order to establish finiteness.

Example 5.2 also demonstrates the importance of *inclusion dependencies* (abbr., IND). These are statements of the form $r[\vec{X}] \subseteq q[\vec{Y}]$, where \vec{X} and \vec{Y} are lists of attributes of the same length (each list may have repeated attributes) and r, q are predicate symbols. A pair $\overline{r}, \overline{q}$ of relations over predicates r and q satisfies the above IND if and only if $\overline{r}[\vec{X}] \subseteq \overline{q}[\vec{Y}]$. In the above example, we relied on the fact that the inclusion $\overline{R}_2[V_2, W_2] \subseteq \overline{p}$ holds in the database instance. Section 8 formalizes IND's in a more general context. We have also seen that projections of relations, such as $\overline{R}_2[X_2, Y_2]$ in the example, played a role in our arguments. We deal with projections in Section 7.

Our final observation is that Example 5.2 illustrates the use of constraints that may hold only on certain components of a relation, but not on the relation as a whole. Reasoning about these “partial” constraints was central to our case-analysis of relation \overline{R}_2 . When combined, partial constraints may yield a global FC of the form $\text{Finite}(\dots)$ (or $\text{Finite}(X_2, Y_2)$, in our case-analysis), which is precisely what we need for the finiteness. Partial constraints is the main topic of the next section.

6 Horizontal Decompositions and Partial Constraints

We have already seen most of the issues that need to be addressed in the development of a formal axiomatization for FC's over multiple IDB-predicates. One novel aspect of this development is the special role played by the horizontal structure of infinite recursive predicates. Previous examples show that we may have to reason explicitly about the decomposition of a relation into components. Intuitively, we associate each component with the rule that generates tuples for that component. However, we may also have to reason about the components themselves and, again, they can be decomposed further. In the limit, this may lead to decompositions with an infinite number of components.

We also introduce the notion of a *partial constraint* (abbr., PC), a generalization of FC's that formalizes that aspect of Example 5.2 where we dealt with constraints that hold only over certain parts of a relation. A PC consists of one or more sets of FC's, and (roughly speaking) holds over a decomposition if each component of the decomposition satisfies one of the sets of FC's. That is, we may know that each component satisfies *some* set of FC's, but we may not know which.

By the end of this section we shall have developed axioms for inferring PC's. Subsequent sections

will extend these results to projection and inclusion dependencies. Finally, we shall apply these axioms to the problem of deciding superfiniteness.

Decompositions

We now develop a formal basis for reasoning about constraints on horizontal decompositions of relations. Let $\bar{p}, \bar{p}_1, \bar{p}_2, \dots$ be a (possibly infinite) set of relations of the same arity. It constitutes a *horizontal decomposition* of \bar{p} if $\bar{p} = \bar{p}_1 \cup \bar{p}_2 \cup \dots$. In this case, we shall write $\bar{p} = \bar{p}_1 \mid \bar{p}_2 \mid \dots$.

Note that the number of component-relations in a decomposition may be infinite. A decomposition is *finite* if it has a finite number of components. Also, if $\bar{\pi}$ is a decomposition, $\cup \bar{\pi}$ will stand for the union of all components of $\bar{\pi}$ (*i.e.*, for the very relation decomposed by $\bar{\pi}$).

Since decompositions are *sets* of relations, the order in which these relations are listed is immaterial. Likewise, we do not require all components of a decomposition to be disjoint. Each relation \bar{p} can be identified with a *singleton* decomposition, one in which \bar{p} is the only component. Furthermore, if $\bar{\pi}_1, \bar{\pi}_2, \dots$ are decompositions of various relations defined over the *same predicate*, then we write $\bar{\pi}_1 \mid \bar{\pi}_2 \mid \dots$ to denote a decomposition whose set of components is the union of the sets of components of the $\bar{\pi}_i$'s; this latter decomposition is called the *union* of $\bar{\pi}_1, \bar{\pi}_2, \dots$. Thus, if $\bar{\pi} = \bar{p}_1 \mid \bar{p}_2$ and $\bar{\rho} = \bar{r}_1 \mid \bar{r}_2$, then $\bar{\pi} \mid \bar{\rho}$ is $\bar{p}_1 \mid \bar{p}_2 \mid \bar{r}_1 \mid \bar{r}_2$.

We shall use the following partial order on decompositions: $\bar{\pi} \sqsubseteq \bar{\rho}$ if and only if $\bar{\pi}$ and $\bar{\rho}$ are defined over the *same predicate* and every component of $\bar{\pi}$ is also a component of $\bar{\rho}$.⁷ For instance, if $\bar{\pi} = \bar{\pi}_1 \mid \bar{\pi}_2 \mid \dots$ then $\bar{\pi}_i \sqsubseteq \bar{\pi}$, for all i .

A slightly weaker relationship, denoted $\bar{\pi} \preceq \bar{\rho}$, holds if each component of $\bar{\pi}$ is *contained* in a component of $\bar{\rho}$ (which might be different for different components of $\bar{\pi}$). It is easy to see that “ \preceq ” may be a cyclic relation (*i.e.*, $\bar{\pi} \preceq \bar{\rho} \preceq \bar{\pi}$ is possible for distinct $\bar{\pi}$ and $\bar{\rho}$), so “ \preceq ” is a *quasi-order*, but not a partial order (quasi-orders often also go under the name “pre-orders”). Clearly, $\bar{\pi} \sqsubseteq \bar{\rho}$ implies $\bar{\pi} \preceq \bar{\rho}$, but not vice versa.

Let $\bar{\pi}$ and $\bar{\rho}$ be decompositions over the same predicate. We shall say that $\bar{\rho}$ is an *approximation* of $\bar{\pi}$ if $\bar{\pi} \preceq \bar{\rho}$ and $\cup \bar{\pi} = \cup \bar{\rho}$ (*i.e.*, they are decompositions of the *same relation*).

Partial Constraints

A *partial constraint* (abbr., *PC*) over a predicate \bar{p} is a statement of the form

$$p : (F_1 \mid F_2 \mid \dots \mid F_n) \tag{2}$$

where the F_i 's are sets of FC's over p . The predicate name may be omitted if it is known or immaterial. Unlike decompositions, the number of components in a PC is always finite. We shall view PC's as sets of sets of FC's, so duplicate occurrences of their components (which are sets of FC's) are discarded and the order in which they are listed is not important. A set F of FC's can be identified with a *singleton partial constraint*—one where F is the only component.

⁷In fact, if we view $\bar{\pi}$ and $\bar{\rho}$ as sets of relations over the same predicate, then $\bar{\pi} \sqsubseteq \bar{\rho}$ if and only if $\bar{\pi} \subseteq \bar{\rho}$.

Note: since predicates have finite arities, *there can be only a finite number of distinct FC's and PC's over a predicate.*

We use the following notation. If α, \dots, γ are PC's over the same predicate then $\alpha \mid \dots \mid \gamma$ is another PC whose set of components is obtained by pulling together the sets of components of α, \dots, γ . For instance, if $\alpha = p : (F_1 \mid F_2)$ and $\beta = p : (F_2 \mid F_3 \mid F_4)$ then $\alpha \mid \beta = p : (F_1 \mid F_2 \mid F_3 \mid F_4)$.

A PC of the above form (2) *holds* in (or is *satisfied* by) a decomposition π if π has a *finite approximation* $\bar{p} = \bar{r}_1 \mid \dots \mid \bar{r}_n$ such that *every* \bar{r}_i satisfies *some* F_j .

The following lemma follows directly from the definition of PC satisfaction and is important for our subsequent study of PC's. It says that there is no need to use approximations to verify satisfaction of PC's in *finite* decompositions.

Lemma 6.1 *Let π be a finite decomposition and α be a PC of the form $p : (F_1 \mid F_2 \mid \dots \mid F_n)$. Then α holds in π if and only if every $\bar{p}_i \in \pi$ satisfies some F_j .*

For infinite decompositions, however, the use of finite approximations is crucial for the definition to make sense. Indeed, suppose we defined satisfaction of (2) on an infinite decomposition $\bar{\pi}$ of relation \bar{p} by requiring every component of $\bar{\pi}$ to satisfy some F_i . In this case, even if every F_i had the form $p : \text{Finite}(1)$ it would still be impossible to conclude that \bar{p} satisfies $p : \text{Finite}(1)$. Indeed, even if \bar{p} 's first attribute had only a finite number of values in every component of $\bar{\pi}$, this attribute may still have an infinite number of values in \bar{p} , as the number of components in $\bar{\pi}$ may be infinite.

We also note that partial constraints are quite different from conditional functional dependencies studied in [10, 9], although both classes of constraints are intended to deal with problems arising when relations have non-uniform horizontal structure.

Entailment of FC's

We say that a set Γ of partial constraints *logically implies* (or *entails*) another partial constraint β , denoted $\Gamma \models \beta$, if β holds in every decomposition in which each PC of Γ does. The next lemma presents several important properties of decompositions and PC's.

We have already defined the union of an arbitrary number of decompositions. We will also need other operators, such as intersection. Let $\bar{\pi}_1, \bar{\pi}_2, \dots$ be decompositions defined over the same predicate. Their *intersection* is a decomposition $\bar{\pi}$ whose components are all relations of the form:

$$\bar{p}_j = \bigcap_{i=1}^{\infty} \bar{p}_{j_i}$$

where $\bar{p}_{j_i} \in \bar{\pi}_i$, for $i = 1, 2, \dots$. Note that each \bar{p}_{j_i} here is a relation that belongs to $\bar{\pi}_i$ and, to produce each component \bar{p}_j of $\bar{\pi}$, exactly one relation is chosen from each $\bar{\pi}_i$. In other words, $\bar{\pi}$ is a component-wise intersection of the decompositions $\bar{\pi}_1, \bar{\pi}_2$, etc. Clearly, $\bar{\pi} \preceq \bar{\pi}_i$.

Lemma 6.2 *1. If $\bar{p} = \bigcap_{i=1}^{\infty} \bar{p}_i$, where $\bar{p}_1, \bar{p}_2, \dots$ are approximations of the same decomposition $\bar{\pi}$, then \bar{p} is also an approximation of $\bar{\pi}$. Intersection of a finite number of finite decompositions is a finite decomposition.*

2. Let Γ be a set of PC's satisfied by a possibly infinite decomposition $\bar{\pi}$. Then there is a finite approximation $\bar{\rho}$ of $\bar{\pi}$ such that $\bar{\rho}$ satisfies Γ .⁸
3. Let $\alpha = F_1 \mid \cdots \mid F_n$ and $\beta = G_1 \mid \cdots \mid G_m$. Then $\alpha \models \beta$ if and only if for every component F_j of α there is a component G_i of β such that $F_j \models G_i$.
4. Let $\alpha = F_1 \mid \cdots \mid F_n$, and $\beta = G_1 \mid \cdots \mid G_m$. Let γ be the PC $F_1 \cup G_1 \mid \cdots \mid F_i \cup G_j \mid \cdots \mid F_n \cup G_m$ (i.e., γ 's components are all possible binary unions of the components of α and β). Then the set $\{\alpha, \beta\}$ is equivalent to γ , i.e., $\{\alpha, \beta\} \models \gamma$, $\gamma \models \alpha$, and $\gamma \models \beta$.

Proof: 1. First, observe that $\bar{\pi}$ and $\bar{\rho}$ are decompositions of the same relation, which is a direct consequence of the definitions of intersection and approximation.

For every component $\bar{p} \in \bar{\pi}$ and each $i = 1, 2, \dots$, there is $\bar{r}_i \in \bar{\rho}_i$ such that $\bar{p} \subseteq \bar{r}_i$, by the definition of approximations. Thus, $\bar{p} \subseteq \bigcap_{i=0}^{\infty} \bar{r}_i$, and the latter intersection is a component of $\bar{\rho}$. Therefore, $\bar{\rho}$ approximates $\bar{\pi}$. The second part of the claim is obvious.

2. Recall that there exists only a finite number of different FC's and PC's over any predicate. Since $\bar{\pi}$ satisfies Γ , there is a finite set of finite approximations of $\bar{\pi}$, say, $\bar{\rho}_1, \dots, \bar{\rho}_k$, such that each PC in Γ is satisfied by some $\bar{\rho}_i$. Let $\bar{\rho}$ be the intersection of $\bar{\rho}_1, \dots, \bar{\rho}_k$. By Claim 1, $\bar{\rho}$ is a finite approximation of $\bar{\pi}$. Since $\bar{\rho} \preceq \bar{\rho}_i$, for all i , it follows from the definition of satisfaction that every PC in Γ holds true in $\bar{\rho}$. Hence, $\bar{\rho}$ is the desired decomposition.

3. The “if”-direction is a simple consequence of the definitions. For the “only if”-part, let $\alpha \models \beta$. Suppose to the contrary that, say, F_1 is such that, for all $j = 1, \dots, m$, $F_1 \not\models G_j$. We will construct a relation that satisfies F_1 but none of the G_j 's. Since this relation, considered as a decomposition, satisfies α but violates β , this will prove Claim 3. For each $j = 1, \dots, m$, let \bar{g}_j be a relation that satisfies F_1 , but violates G_j (\bar{g}_j must exist since $F_1 \not\models G_j$). Then, $\bar{g} = \bar{g}_1 \cup \dots \cup \bar{g}_m$ is the desired relation that satisfies F_1 but none of the G_j 's.

4. By Claim 2, it suffices to consider finite decompositions only. Let $\bar{\pi} = \bar{p}_1 \mid \cdots \mid \bar{p}_k$ be some decomposition that satisfies α and β . By definition, every \bar{p}_j satisfies some F_{k_j} and G_{l_j} . In particular, \bar{p}_j satisfies $F_{k_j} \cup G_{l_j}$, which is one of the components of γ . Thus, every component of $\bar{\pi}$ satisfies some component of γ . Hence γ holds in $\bar{\pi}$, which proves the first entailment of Claim 4. The last two entailments follow from Claim 3, since $F_i \cup G_j \models F_i$ and $F_i \cup G_j \models G_j$. \square

Claim 2 of Lemma 6.2 lets us limit our consideration to finite decompositions as far as PC-implication is concerned. Infinite decompositions will enter the picture only in the proof of completeness of our inference system, namely, in Proposition 9.1. Furthermore, by Claim 4 of Lemma 6.2, sets of PC's can always be replaced by a single, equivalent PC. This lemma, thus, suggests the following inference rules for PC's:

PC-Rules: Let $\alpha = F_1 \mid \cdots \mid F_n$, $\beta = G_1 \mid \cdots \mid G_m$, and $\gamma = F_1 \cup G_1 \mid \cdots \mid F_i \cup G_j \mid \cdots \mid F_n \cup G_m$. Then:

- (i) **from** α and $\{F_i \vdash G_k \mid i = 1, \dots, n, 1 \leq k \leq m\}$ **infer** β

⁸The point here is that all PC's in Γ must hold in the same finite decomposition $\bar{\rho}$. The definition of satisfaction ensures only that each $\gamma \in \Gamma$ holds in *some* approximation of π , which may be different for different γ .

(ii) **from α and β infer γ**

Note that the inference rule (i) is well-formed, since the provability relation “ \vdash ” used there is provability with respect to the FC-rules only; it was introduced in Section 4.

Let Γ be a set of PC’s and γ be a PC. We shall write $\Gamma \vdash \gamma$ to mean that γ can be derived from Γ using PC-rules above (and FC-rules, as needed).

Proposition 6.3 *Let all integrity constraints be PC’s. Then the set of PC-rules and FC-rules is sound and complete for inferring PC’s: $\Gamma \models \gamma$ if and only if $\Gamma \vdash \gamma$.*

Proof: Soundness follows from Lemma 6.2. For completeness, suppose that $\Gamma \models \gamma$. By the second PC-inference rule, we can replace Γ by a single PC, α , which, by Claim 4 of Lemma 6.2, is equivalent to Γ , *i.e.*, $\Gamma \models \alpha$ and $\alpha \models \Gamma$. Thus, $\alpha \models \gamma$. Let α be $F_1 \mid \cdots \mid F_n$ and γ be $G_1 \mid \cdots \mid G_m$. By Claim 3 of Lemma 6.2, for every F_i there is G_k such that $F_i \models G_k$. Now, by Proposition 4.2, the FC-rules are complete and so $F_i \vdash G_k$. Therefore, γ follows from α by the first PC-rule. Thus, $\Gamma \vdash \alpha \vdash \gamma$. \square

Corollary 6.4 *Implication and equivalence of sets of PC’s is decidable.*

Proof: By the second PC inference rule, it suffices to consider single PC’s rather than sets. Let α and β be PCs such that $\alpha \models \beta$. By Proposition 6.3, $\alpha \vdash \beta$. From the proof of Proposition 6.3, it follows that since α and β are singleton PC’s, β can be derived from α solely via the first PC-rule. Thus, the problem reduces to the implication problem for sets of FC’s. This is decidable, by Corollary 4.3.

The algorithm takes time $O(n^4)$ in the size of α and β . Indeed, checking $F \models G$ takes quadratic time, and we have to do this for each $F \in \alpha$ and $G \in \beta$. For sets of PC’s, the complexity of this algorithm is exponential, since it takes $O(n^m)$ steps to convert a set of m PC’s with n components each into a single PC. \square

7 Projections of Dependencies

In the sequel, we will be dealing with inter-relational constraints such as projection and inclusion dependencies. The notion of satisfaction for these dependencies has to be extended, so it will apply to decompositions. We also need to extend the notion of database instance, so that decompositions could be used in place of relations. Let p_1, \dots, p_k be predicate symbols. An *extended database instance* is a mapping that assigns a decomposition (rather than a relation) to each p_j .

From the examples we already know that projections of relations may get involved in reasoning about finiteness. This is further illustrated by the following example.

Example 7.1 (Use of projected dependencies) Let \mathbf{P} be the following IDB:

$$\begin{aligned}
R_1 &: s(X_1, Y_1) \leftarrow f(X_1, Y_1, Z_1) \\
R_2 &: q(X_2) \leftarrow s(X_2, Y_2), b(Y_2, Z_2) \\
b &: \text{Finite}(1) \\
f &: 2 \dashrightarrow 1
\end{aligned}$$

Then the query $?- q(X_3)$ is superfinite. Indeed, let M be a fixpoint model of P . For each value in the second attribute of \bar{s} (the relation assigned to s by M) there is only a finite number of corresponding values in the first attribute, due to the FC. When \bar{s} is used in the second rule, its second argument is bound to a finite set of values by $b(Y_2, Z_2)$. Hence, the first argument of \bar{s} is also bound to a finite set. \square

Now, let us see how inference rules for constraints can help us accomplish the above reasoning automatically. Let M be a fixpoint model of \mathbf{P} and let \bar{R}_1, \bar{R}_2 be the relations corresponding to R_1 and R_2 , respectively. As in Examples 5.1 and 5.2, these relations are defined as follows:

$$\begin{aligned}
\bar{R}_1(X_1, Y_1, Z_1) &\stackrel{\text{def}}{=} \{ \langle x, y, z \rangle \mid f(x, y, z) \in M \} \\
\bar{R}_2(X_2, Y_2, Z_2) &\stackrel{\text{def}}{=} \{ \langle x, y, z \rangle \mid s(x, y), b(y, z) \in M \}.
\end{aligned}$$

The only FC's given initially are of the form $R_2 : \text{Finite}(Y_2)$ and $R_1 : Y_1 \dashrightarrow X_1$. By themselves, these constraints are insufficient to ensure finiteness of q . However, we observe that the inclusion $\bar{R}_2[X_2, Y_2] \subseteq \bar{R}_1[X_1, Y_1]$ holds in our database instance. Because of this inclusion, the FC $Y_1 \dashrightarrow X_1$ on \bar{R}_1 induces the same FC on $\bar{R}_1[X_1, Y_1]$, which in turn induces the FC $Y_2 \dashrightarrow X_2$ on $\bar{R}_2[X_2, Y_2]$. Now, since Y_2 is a finite argument (by the initial constraints), we conclude (using the second auxiliary rule for FC's) that X_2 (hence q) is finite.

In this reasoning, the first step was to derive an FC on the projected relation $\bar{R}_1[X_1, Y_1]$ from the FC's on \bar{R}_1 . In the second step, we used an inclusion dependency to further the derivation process. Each of these steps, although simple, is necessary for rigorous axiomatization. (In general, projecting FC's may be somewhat more involved because of the possible repeated arguments in projection lists.)

In order to perform the above reasoning proof-theoretically, we have to look at the properties of projections more closely. First, we define projection dependencies for ordinary databases. Let p be an n -ary predicate symbol, and let \vec{X} be a sequence of its attributes. Let r be a predicate symbol whose arity equals the length of \vec{X} . A *projection dependency*⁹ (abbr., PRD) is a statement of the form $r = p[\vec{X}]$. It holds in an (ordinary) database instance if and only if $\bar{r} = \bar{p}[\vec{X}]$, where \bar{r} and \bar{p} are relations assigned to r and p by that database instance.

The position-number notation is often convenient for dealing with projections. For instance, we can write $p[3, 2, 2]$ to denote the projection of p on the third and then twice the second attribute.

Let $r = p[i_1, \dots, i_m]$ be a PRD in the position-number notation. *Associated* with this PRD is a natural *attribute mapping* τ from the attributes of r to those of p . It is defined as follows: $\tau(r : j) = p : i_j$, for $j = 1, \dots, m$. Thus, for $r = p[3, 2, 2]$, the associated attribute mapping is $\{r : 1 \mapsto p : 3, r : 2 \mapsto p : 2, r : 3 \mapsto p : 2\}$. We extend this mapping to sets and sequences of attributes in the usual way.

⁹Our notion of projection dependencies should not be confused with that of [13], which is a different concept.

Consider a PRD $r = p[\vec{X}]$, and let τ be the associated mapping. A *projection* of $p : Y \dashrightarrow Z$ on r (or on \vec{X}) is any FC of the form $r : Y' \dashrightarrow Z'$, where $\tau(Y') = Y$ and $\tau(Z') = Z \cap X$ (here X is \vec{X} viewed as a set). There may exist several such Y' and Z' , since τ does not have to be a 1-1 mapping; therefore, an FC may have several projections.

For the above PRD $r = p[3, 2, 2]$ and the FC $p : 2 \dashrightarrow \{1, 3\}$, the projections of this FC on r are $r : 2 \dashrightarrow 1$, $r : 3 \dashrightarrow 1$, and $r : \{2, 3\} \dashrightarrow 1$ (because both $r : 2$ and $r : 3$ are mapped to $p : 2$ by the mapping associated with the PRD).

Note that if $Y \not\subseteq X$ then $Y \dashrightarrow Z$ has no projection on \vec{X} . For convenience, in this case we assume that the projection is some trivial FC over r , e.g., $\{\} \dashrightarrow \{\}$. It is easy to see from the definition that for an FC of the form $\text{Finite}(Y)$ one of the projections on \vec{X} is $\text{Finite}(\tau^{-1}(Y \cap X))$, where $\tau^{-1}(Y \cap X)$ is a prototype (or pre-image) of $Y \cap X$ under τ .

It is also convenient to define a *promotion* operation that works in the direction opposite to FC-projection: in the above notation, if f' is an FC $Y' \dashrightarrow Z'$ over r then $\tau(Y') \dashrightarrow \tau(Z')$ is an FC over p . We denote this latter FC by $\tau(f')$ and call it a *promotion* of f' . Note that f' is an \vec{X} -projection of $\tau(f')$, but there may be other projections, too.

Consider a set F of FC's on p . We define its *projection* on \vec{X} as follows: $F[\vec{X}] = \{f' \mid f' \text{ is an } \vec{X}\text{-projection of some } f \in F^+\}$, where F^+ is the closure of F with respect to FC-rules. *Promotion* of F is defined as follows: $\tau(F) = \{\tau(f) \mid f \in F\}$.

If F is a set of FC's that holds in \bar{p} then, by Lemma 7.2, $F[\vec{X}]$ is exactly the set of constraints guaranteed to hold in \bar{r} . It should be noted that projections of trivial constraints may be non-trivial. For example, consider the PRD $r = p[1, 1]$. Then $r : 1 \dashrightarrow 2$ and $r : 2 \dashrightarrow 1$ both are nontrivial projections of the trivial FC $p : 1 \dashrightarrow 1$.

Lemma 7.2 *Let \bar{r} and \bar{p} be relations such that $\bar{r} = \bar{p}[\vec{X}]$, where \vec{X} is a sequence of attributes of \bar{p} .*

1. *If δ is an FC that holds in \bar{p} , then all its projections on \vec{X} hold in \bar{r} .*
2. *If δ' is an FC that holds in \bar{r} , then its promotion $\tau(\delta')$ holds in \bar{p} .*
3. *If $F \models \tau(\delta')$ then $F[\vec{X}] \models \delta'$.*

Proof: Claims 1 and 2 follow straight from the definitions. For Claim 3, $\tau(\delta') \in F^+$ (due to the completeness of FC inference), hence $\delta' \in F^+[\vec{X}]$ (since δ' is a projection of $\tau(\delta')$). Therefore, $F^+[\vec{X}] \models \delta'$. \square

We now extend the above definitions and Lemma 7.2 to cover PC's and horizontal decompositions.

Let $\bar{\pi} = \bar{p}_1 \mid \bar{p}_2 \mid \dots$ be a decomposition of a relation \bar{p} , and let \vec{X} be a sequence of attributes of \bar{p} . Then the *projection* of $\bar{\pi}$ on \vec{X} , denoted $\bar{\pi}[\vec{X}]$, is $\bar{p}_1[\vec{X}] \mid \bar{p}_2[\vec{X}] \mid \dots$. If α is a PC over p of the form $F_1 \mid \dots \mid F_n$, then its *projection* on \vec{X} , denoted $\bar{\alpha}[\vec{X}]$, is defined as $F_1[\vec{X}] \mid \dots \mid F_n[\vec{X}]$.

As with FC's, the *promotion* operation takes PC's over r and produces PC's over p : if β is a PC of the form $r : F'_1 \mid \dots \mid F'_n$ and τ is the attribute mapping associated with the PRD $r = p[\vec{X}]$, then $\tau(\beta)$ is $p : \tau(F'_1) \mid \dots \mid \tau(F'_n)$.

The notion of satisfaction for projection dependencies can be naturally generalized from ordinary database instances to extended instances. We say that a PRD of the form $r = p[\vec{X}]$ is *satisfied* in an extended database instance $\overline{\mathbf{D}}$ if $\overline{\pi}_r = \overline{\pi}_p[\vec{X}]$, where $\overline{\pi}_r$ and $\overline{\pi}_p$ are decompositions that $\overline{\mathbf{D}}$ assigns to r and p , respectively.

We should note that PRD's are not indispensable in our axiomatization. However, in our setting, they help simplify definitions and inference rules.

Corollary 7.3 *Let $r = p[\vec{X}]$ be a PRD with an associated attribute mapping τ , and let $\overline{\rho}$ and $\overline{\pi}$ be decompositions over r and p such that $\overline{\rho} = \overline{\pi}[\vec{X}]$. Then:*

1. *If a PC α holds in $\overline{\pi}$ then $\alpha[\vec{X}]$ holds in $\overline{\rho}$.*
2. *If a PC α' holds in $\overline{\rho}$, then its promotion $\tau(\alpha')$ holds in $\overline{\pi}$.*
3. *If $\alpha \models \tau(\beta')$, where α is a PC over p and β' is a PC over r , then $\alpha[\vec{X}] \models \beta'$.*

Proof: **1.** Let $\alpha = F_1 \mid \dots \mid F_k$. Since α holds in $\overline{\pi}$, the latter must have a finite approximation $\overline{\pi}' = \overline{p}'_1 \mid \dots \mid \overline{p}'_n$, where every \overline{p}'_i satisfies some F_{j_i} .

Clearly, $\overline{\pi}'[\vec{X}]$ is a finite approximation of $\overline{\pi}[\vec{X}] (= \overline{\rho})$. Therefore, we only need to show that $\overline{\pi}'[\vec{X}]$ satisfies $\alpha[\vec{X}] = F_1[\vec{X}] \mid \dots \mid F_k[\vec{X}]$. Consider some $\overline{p}'_i[\vec{X}]$. Since \overline{p}'_i satisfies F_{j_i} , it also satisfies $F_{j_i}^+$. By Claim 1 of Lemma 7.2, every FC in $F_{j_i}[\vec{X}]$ is satisfied by $\overline{p}'_i[\vec{X}]$. This shows that every component of $\overline{\pi}'[\vec{X}]$ satisfies some component of $\alpha[\vec{X}]$, which proves part 1.

2. Let $\overline{\rho}' = \overline{r}'_1 \mid \dots \mid \overline{r}'_n$ be a finite approximation of $\overline{\rho}$ that satisfies $\alpha' = F'_1 \mid \dots \mid F'_k$. We construct $\overline{\pi}' = \overline{p}'_1 \mid \dots \mid \overline{p}'_n$, a finite approximation of $\overline{\pi}$, such that $\overline{\pi}'[\vec{X}] = \overline{\rho}'$. Namely, we define $\overline{p}'_i = \{t \mid t \in \cup \overline{\pi} \text{ and } t[\vec{X}] \in \overline{r}'_i\}$, $i = 1, \dots, n$. In particular, both $\overline{\pi}$ and $\overline{\pi}'$ are decompositions over p .

Since $(\cup \overline{\pi})[\vec{X}] \supseteq \overline{r}'_i$, $i = 1, \dots, n$, we have $\overline{p}'_i[\vec{X}] = \overline{r}'_i$, *i.e.*, $\overline{\pi}'[\vec{X}] = \overline{\rho}'$. Since $\cup \overline{\rho}' = \cup \overline{\rho} = (\cup \overline{\pi})[\vec{X}]$ by the assumptions, it follows that $\cup \overline{\pi}' = \{t \mid t \in \cup \overline{\pi}\} = \cup \overline{\pi}$. To see that $\overline{\pi}'$ approximates $\overline{\pi}$, it remains to be shown that for each $\overline{p} \in \overline{\pi}$, we have $\overline{p} \subseteq \overline{p}'_i$, for some $\overline{p}'_i \in \overline{\pi}'$. Let \overline{p} be a component of $\overline{\pi}$. Then $\overline{p}[\vec{X}]$ is a component of $\overline{\rho}$. Since $\overline{\rho}'$ approximates $\overline{\rho}$, it follows that $\overline{p}[\vec{X}]$ is a subset of some \overline{r}'_i . Therefore, $\overline{p} \subseteq \{t \mid t \in \cup \overline{\pi} \text{ and } t[\vec{X}] \in \overline{r}'_i\} = \overline{p}'_i$ (the latter holds by construction of \overline{p}'_i), *i.e.*, $\overline{p} \subseteq \overline{p}'_i$.

Finally, we need to show that $\overline{\pi}'$ satisfies $\tau(\alpha')$. Let $\alpha' = F'_1 \mid \dots \mid F'_k$. Consider some \overline{p}'_i and the associated equation $\overline{r}'_i = \overline{p}'_i[\vec{X}]$. Since $\overline{\rho}'$ satisfies α' , \overline{r}'_i satisfies some F'_j . Hence, by Claim 2 of Lemma 7.2, \overline{p}'_i satisfies $\tau(F'_j)$. Since \overline{p}'_i was chosen arbitrarily, we conclude that $\overline{\pi}'$ satisfies $\tau(\alpha')$.

3. Let $\alpha = F_1 \mid \dots \mid F_k$ and $\beta' = G'_1 \mid \dots \mid G'_n$. Since $\alpha \models \tau(\beta')$, every F_i has some G'_{j_i} such that $F_i \models \tau(G'_{j_i})$ (Lemma 6.2, Claim 3). By Lemma 7.2, Claim 3, $F_i[\vec{X}] \models G'_{j_i}$. Hence, every component of $\alpha[\vec{X}]$ entails some component of β' , *i.e.*, $\alpha[\vec{X}] \models \beta'$. \square

Corollary 7.3 suggests the following derivation rules for constraints in projections of horizontal decompositions:

PRD-Rules: Let $r = p[\vec{X}]$ be a PRD with an associated attribute mapping τ . Suppose also that α and β are PC's over p and r , respectively. Then:

- (i) **from** $p : \alpha$ and $r = p[\vec{X}]$ **infer** $r : \alpha[\vec{X}]$.

(ii) **from** $r : \beta$ and $r = p[\vec{X}]$ **infer** $p : \tau(\beta)$.

Soundness of these rules follows from Corollary 7.3. We shall see later that these rules are also complete for inferring PC's.

We will often need to extend decompositions over $p[\vec{X}]$ that satisfy a set $\Gamma[\vec{X}]$ of PC's to decompositions over p that satisfy Γ . In general, this cannot be done for every possible decomposition, which is consistent with the corresponding result about functional dependencies [34]. Fortunately, we only need to extend a very special class of decompositions, called *simple* decompositions.

Let \bar{p} be a relation for p and let F be a set of FC's on p . A relation \bar{p} is called *F-simple* if all of the following conditions are satisfied:

- The components of the tuples in \bar{p} are drawn from a fixed infinite set $\{\mathbf{a}, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \dots\}$ of constants, where \mathbf{a} is the “distinguished” element.
- Each tuple in \bar{p} can use at most two constants: one of the \mathbf{b}_i 's and \mathbf{a} . Either \mathbf{a} or \mathbf{b}_i may be missing, but two different \mathbf{b}_i 's cannot appear in the same tuple.
- For each tuple $t \in \bar{p}$, the set A_t of all attributes where t assumes the distinguished value \mathbf{a} is F -closed, *i.e.*, $(A_t)_F^+ = A_t$. (The closure, X_F^+ , was defined in Section 4 to mean $\{B \mid F \models X \twoheadrightarrow B\}$.)

Note that the relation constructed in Proposition 4.2 is F -simple. In the theory of FC's, F -simple relations play the role analogous to that of the 2-tuple relations in the theory of functional and multivalued dependencies (*cf.* [37]). So far, the properties of FC's were similar to those of functional dependencies—even the inference rules were the same. The next lemma (Claims 2 and 4) shows that FC's have certain properties that do not hold for FD's, even in the world of 2-tuple relations.¹⁰

Lemma 7.4

1. **Satisfaction:** *Each F-simple relation satisfies F.*
2. **Union:** *Union of any (even infinite) number of F-simple relations is F-simple (and, by Claim 1, satisfies F).*
3. **Projection:** *If \bar{p} is F-simple then $\bar{p}[\vec{X}]$ is $F[\vec{X}]$ -simple.*
4. **Expansion:** *Let $r = p[\vec{X}]$ be a PRD, F be a set of FC's on p , and \bar{r} be an $F[\vec{X}]$ -simple relation on r . Then there is an F-simple relation \bar{p} on p such that $\bar{r} = \bar{p}[\vec{X}]$.*
5. **Relaxation:** *If $F \models G$ then every F-simple relation is also G-simple.*

Proof: 1. Let $Y \twoheadrightarrow B \in F$, and t_1, t_2, \dots be a set of tuples all of which agree on Y . We have to show that among $t_1[B], t_2[B], \dots$ there is only a finite number of different values. If, for some

¹⁰In fact, if Claims 2 and 4 were true for FD's, we could have used the techniques of Proposition 9.1, below, to establish a completeness result for FD's and inclusion dependencies, which is impossible according to [5].

$C \in Y$ and \mathbf{b}_k , we have $t_1[C] = t_2[C] = \dots = \mathbf{b}_k$, then, by the definition of F -simplicity, for all i , the value $t_i[B]$ can be either \mathbf{b}_k or \mathbf{a} , in which case we are done.

If such an attribute C does not exist then $t_i[Y] = \langle \mathbf{a}, \mathbf{a}, \dots, \mathbf{a} \rangle$, for all i . But then $Y \subseteq A_{t_i}$, for all i , and, since each A_{t_i} is F -closed, it follows that $B \in A_{t_i}$, for all i . Therefore, $\{t_i[B] \mid i = 1, 2, 3, \dots\} = \{\mathbf{a}\}$.

Note that our last argument applies also in case $Y = \{\}$, *i.e.*, for constraints of the form $\text{Finite}(B)$. Therefore, if $\text{Finite}(B) \in F$ then all tuples have the value \mathbf{a} on B .

2. Follows from the definition of F -simple relations.

3. Of the three conditions for F -simplicity, only the last one, closedness of the A_t 's is not obvious.

Suppose $F[\vec{X}] \models A_t \dashrightarrow B$. By Lemma 7.2, Claim 2, $F \models \tau(A_t) \dashrightarrow \tau(B)$, where τ is the attribute mapping associated with the projection on \vec{X} . Let $t' \in \bar{p}$ be a tuple such that $t'[\vec{X}] = t$. Clearly, $A_{t'} \supseteq \tau(A_t)$. Since $A_{t'}$ is F -closed, $\tau(B) \in A_{t'}$. Hence, $t'[\tau(B)] = \mathbf{a}$ and thus $t[B] = \mathbf{a}$. Therefore, $B \in A_t$. Since B was chosen arbitrarily, this means that A_t is $F[\vec{X}]$ -closed.

4. We extend each tuple $t \in \bar{r}$ to a tuple t^e over p as follows. Let τ be the attribute mapping associated with the PRD. We construct t^e so that it has the value \mathbf{a} over $(\tau(A_t))_F^+$; over the other attributes, we patch t^e by letting it assume the very same value \mathbf{b}_i that appears in t . If no \mathbf{b}_i appears in t , use \mathbf{a} to patch t^e .

Let $\bar{p} = \{t^e \mid t \in \bar{r}\}$. By construction, \bar{p} is F -simple. We only need to check that for every $t \in \bar{r}$ and every attribute B of r , $t[B] = t^e[\tau(B)]$.

If $B \in A_t$, this property is obvious. If $B \notin A_t$, then $t[B] = \mathbf{b}_i$, for some i , and $\alpha = A_t \dashrightarrow B$ is not implied by $F[\vec{X}]$. But then $\alpha' = \tau(A_t) \dashrightarrow \tau(B) = \tau(\alpha)$ is not implied by F , by Lemma 7.2, Claim 3. Hence, $\tau(B) \notin (\tau(A_t))_F^+$ and, by construction, $t^e[\tau(B)] = \mathbf{b}_i = t[B]$.

5. We only need to remark that every F -closed set of attributes is also G -closed. \square

Claim 2 of Lemma 7.4 does not hold for arbitrary relations, since taking union of an infinite number of relations may lead to the violation of some of the FC's. Claim 4 is not true, in general, since not every relation can be extended from $p[\vec{X}]$ to p so that the constraints on p will hold.

Lemma 7.4 is extended to horizontal decompositions and PC's as follows. Let $\bar{\pi} = \bar{p}_1 \mid \bar{p}_2 \mid \dots$ be a decomposition over p and let $\alpha = F_1 \mid \dots \mid F_k$ be a PC defined on p . We say that $\bar{\pi}$ is α -simple if each relation in $\bar{\pi}$ is F_j -simple for some F_j .

Notice that if $\bar{\pi}$ is α -simple then it has a finite α -simple approximation; hence $\bar{\pi}$ satisfies α . This approximation has the form $\bar{r}_1 \mid \dots \mid \bar{r}_k$ (note: it has the same number of components as α), where each \bar{r}_i is the union of all those $\bar{p}_i \in \bar{\pi}$ that are F_i -simple. By Claim 2 of Lemma 7.4, each \bar{r}_i is F_i -simple. Hence, every α -simple decomposition satisfies α .

We say that $\bar{\pi}$ is Γ -simple if and only if it is γ -simple for some single PC that is equivalent to Γ (such a PC exists by Claim 4 of Lemma 6.1). By Claim 5 of the next corollary, Γ -simplicity does not depend on the specific choice of this PC.

Corollary 7.5 *Let Γ be a set of PC's over predicate p . Then:*

1. Every Γ -simple decomposition satisfies Γ .

2. Union¹¹ of any number of Γ -simple decompositions is Γ -simple.
3. For any Γ -simple decomposition, its projection on \vec{X} is $\Gamma[\vec{X}]$ -simple.
4. Let $r = p[\vec{X}]$ be a PRD, where r and p are predicates. Let $\bar{\pi}_r$ be a $\Gamma[\vec{X}]$ -simple decomposition over r . Then there is a Γ -simple decomposition $\bar{\pi}_p$ over p such that $\bar{\pi}_r = \bar{\pi}_p[\vec{X}]$.
5. If $\Gamma \models \Delta$, where Δ is a set of PC's, then every Γ -simple decomposition is also Δ -simple. Γ -simplicity does not depend on the specific choice of a PC to replace Γ (in the definition of Γ -simplicity)
6. If $\Gamma \not\models \Delta$ then there is a Γ -simple decomposition $\bar{\pi}$ such that $\bar{\pi} \models \Gamma$ and $\bar{\pi} \not\models \Delta$.

Proof: The proof is carried out by first reducing the problem to a single PC, then considering individual component of the decompositions, and then appealing to Lemma 7.4. Claim 1 was verified just above the statement of this lemma. We will skip the simple proofs of Claims 2, 3, and 4, and present proofs for Claims 5 and 6.

Claim 5: Let γ be a PC equivalent to Γ such that $\bar{\pi}$ is γ -simple and let δ be a PC equivalent to Δ . Since $\Gamma \models \Delta$, it follows that $\gamma \models \delta$.

Let $\gamma = G_1 \mid \dots \mid G_n$ and $\delta = D_1 \mid \dots \mid D_m$, where the G_i and D_j are sets of FC's. By definition, every relation $\bar{r}_k \in \bar{\pi}$ is G_{i_k} -simple, for some $G_{i_k} \in \gamma$. Since $\gamma \models \delta$, Claim 3 of Lemma 6.2 ensures that there is $D_{j_k} \in \delta$ such that $G_{i_k} \models D_{j_k}$. Hence, by Claim 5 of Lemma 7.4, \bar{r}_k is D_{j_k} -simple. In other words, every relation in $\bar{\pi}$ is simple with respect to one of the components of δ ; hence, $\bar{\pi}$ is δ -simple and thus Δ -simple.

It follows from here that γ -simplicity (and thus Γ -simplicity) is invariant with respect to PC-equivalence. Moreover, Γ -simplicity does not depend on the PC chosen to represent Γ . Indeed, in the above proof, δ is an arbitrary PC equivalent to Δ and we have shown that every Γ -simple decomposition is δ -simple. If we now take $\Delta = \Gamma$, the claim follows.

Claim 6: Let $\gamma = G_1 \mid \dots \mid G_n$ and $\delta = D_1 \mid \dots \mid D_m$ be PC's that are equivalent to Γ and Δ , respectively. Since $\gamma \not\models \delta$, we must have $G_i \not\models D_j$, for some i and each $j = 1, \dots, m$. By Proposition 4.2, there must exist a relation \bar{r} , such that $\bar{r} \models G_i$ and $\bar{r} \not\models D_j$, for all $j = 1, \dots, m$.

By examining the construction used in Proposition 4.2, one can easily see that the above relation \bar{r} , if constructed according to the recipe in that proposition, is G_i -simple. Therefore, the decomposition $\bar{\pi}$ that has \bar{r} as its only component is γ -simple, so it satisfies γ . However, since \bar{r} violates all D_j 's, it follows that $\bar{\pi}$ violates δ . \square

8 Inclusion and Decomposition Dependencies

Let p and r be predicates of the same arity. An *inclusion dependency* (IND) is a statement of the form $r \subseteq p$. Let $\bar{\mathbf{D}}$ be a (extended) database instance that assigns decompositions $\bar{\pi}_p$ and $\bar{\pi}_r$ to p and

¹¹Union of decompositions, defined in Section 6, should not be confused with the union of relations mentioned in Lemma 7.4, Claim 2. The former is the union of sets of relations, while the latter is the union of relations (*i.e.*, of sets of tuples).

r , respectively. We say that the IND $r \subseteq p$ holds in $\overline{\mathbf{D}}$ if and only if $\overline{\pi}_r \sqsubseteq \overline{\pi}_p$ (i.e., $\overline{\pi}_p$ includes all components of $\overline{\pi}_r$).

Let $r \subseteq p$ be an IND, where r and p are predicates of the same arity. Let $p : \alpha$ be a PC on p expressed in the position-number notation (e.g., $p : 1 \dashrightarrow 2$). Then r “inherits” the PC $r : \alpha$ (i.e., the same PC α , but with p replaced by r ; for instance, $p : 1 \dashrightarrow 2$ becomes $r : 1 \dashrightarrow 2$).

Finally, we need one more class of dependencies called *decomposition dependencies*. This dependency arises when a predicate is defined by several rules, so every decomposition of the corresponding relation becomes a union of the decompositions determined by the rules.

Let p, r_1, r_2, \dots, r_n be predicates of the same arity. A *decomposition dependency* (abbr., DD) is a statement of the form $p = r_1 \mid \dots \mid r_n$. It is satisfied in $\overline{\mathbf{D}}$ if and only if $\overline{\pi}_p = \overline{\pi}_{r_1} \mid \dots \mid \overline{\pi}_{r_n}$, where $\overline{\pi}_p, \overline{\pi}_{r_i}, i = 1, 2, \dots, n$, are decompositions assigned by $\overline{\mathbf{D}}$ to p and r_i , respectively. In plain English, this means that the component-relations comprising $\overline{\pi}_p$ are all and only the components that appear in the decompositions $\overline{\pi}_{r_1}, \dots, \overline{\pi}_{r_n}$.

We are now ready to present the remaining inference rules:

IND-Rules: Let r, p be predicates of the same arity, and $p : \alpha$ be a PC on p . Then

- (i) **from** $r \subseteq p$ and $p : \alpha$ **infer** $r : \alpha$.

DD-Rules: Let p, r_1, \dots, r_n be predicates of the same arity, and $p : \alpha, r_1 : \alpha_1, \dots, r_n : \alpha_n$ be PC's on p, r_1, \dots, r_n , respectively. Then:

- (i) **from** $p = r_1 \mid \dots \mid r_n$ and $r_1 : \alpha_1, \dots, r_n : \alpha_n$ **infer** the PC $p : (\alpha_1 \mid \dots \mid \alpha_n)$ on p .
- (ii) **from** $p = r_1 \mid \dots \mid r_n$ and $p : \alpha$ on p **infer** the PC's $r_i : \alpha$ on r_i , for $i = 1, 2, \dots, n$.

Note that from the definition of DD's and IND's, it follows that $(p = r_1 \mid \dots \mid r_n) \models (r_i \subseteq p)$, for all i . We did not include this as an inference rule, because we are not interested in deriving new IND's.

9 Completeness of the Axiomatization

Proposition 9.1 *Let the set of integrity constraints consist of PC's, PRD's, IND's and DD's. Then the set of PC-rules, PRD-rules, IND-rules, and DD-rules is sound and complete for inferring PC's.*

Proof: Soundness is easy and is left to the reader (only the soundness of IND-rules and DD-rules remains to be verified). To prove completeness, suppose Γ is a set of constraints that includes PC's, PRD's, IND's and DD's.

For every predicate r involved in Γ , let $G(r)$ denote the set of all PC's over r that can be derived from Γ via the inference rules for PC's, PRD's, IND's, and DD's. The PC-rules let us assume that $G(r)$ is, in fact, a single PC. Since each $G(r)$ obviously implies the PC's in Γ originally specified for r , we can assume, without loss of generality, that the only PC's in Γ are the $G(r)$'s.

We shall sometimes write $G(r)$ as $r : G(r)$, which may be necessary to avoid ambiguity. For instance, when we need to say that $G(r)$ also holds over some predicate p other than r , it will be necessary to attach a prefix to distinguish $p : G(r)$ from $r : G(r)$.

The proof of completeness is carried out by contradiction. Suppose α is a PC such that $\Gamma \models \alpha$, but $\Gamma \not\models \alpha$. For concreteness, let us assume that α is specified over predicate q . Then $G(q) \not\models \alpha$. Thus, by Proposition 6.3, there exists a decomposition $\bar{\pi}_q^0$ over q that satisfies $G(q)$, but violates α . Moreover, Claim (6) of Corollary 7.5 ensures that $\bar{\pi}_q^0$ can be chosen to be $G(q)$ -simple. The rest of the predicates in Γ are initially assigned decompositions of empty relations. Let $\bar{\mathbf{D}}^0$ denote this assignment of decompositions to predicates. This extended database instance is our starting point.

A Γ -simple database instance is one that assigns a $G(r)$ -simple decomposition to each predicate r . By Corollary 7.5, Γ -simple database instances satisfy all the PC's in Γ .

By construction, $\bar{\mathbf{D}}^0$ is Γ -simple, so it satisfies the PC's in Γ . Starting with $\bar{\mathbf{D}}^0$, we construct a Γ -simple database instance that, in addition, will satisfy all PRD's, IND's, and DD's in Γ . The construction uses a *chase process* that is akin to that used for inclusion dependencies [15].

The chase process consists in applying *compensating rules* to an intermediate database instance $\bar{\mathbf{D}}$, which initially is identical to $\bar{\mathbf{D}}^0$ but subsequently is changed by rule applications. Given $\bar{\mathbf{D}}$ and a predicate r , we shall use $\bar{\pi}_r$ to denote the decomposition instance that $\bar{\mathbf{D}}$ assigns to r .

A compensating rule is triggered whenever one of the dependencies, an IND, a PRD, or a DD, becomes violated by the database instance. The rule is applied in such a way that the intermediate database instance grows *monotonically*. This means that if $\bar{\mathbf{D}}$ is an extended database instance before the rule application and $\bar{\mathbf{D}}'$ is the instance after the application then $\bar{\pi}_r \sqsubseteq \bar{\pi}_r'$, for every predicate r .

Application of a rule to a Γ -simple instance changes it so that the new instance remains Γ -simple, but the dependency that triggered the compensating rule becomes satisfied. Although being Γ -simple assures that the new instance satisfies the PC's, any rule-application may bring about violation of another PRD, IND, or DD of Γ (that may have been previously satisfied). Hence, another compensating rule may be triggered, and the chase process may go on forever.

Fortunately, this process has a limit, $\bar{\mathbf{D}}^\infty$, that assigns to each predicate r the union of all decompositions that the various intermediate database instances assign to r .

It is not hard to show that all the PC's, IND's, PRD's and DD's are satisfied in $\bar{\mathbf{D}}^\infty$. For the PC's, we note that $\bar{\mathbf{D}}^\infty$ is Γ -simple, since all intermediate database instances are Γ -simple and, by Claim 2 of Corollary 7.5, so must be their union. For any other dependency β , satisfaction follows by examining the following three cases:

- (i) β is an IND $r \subseteq p$;
- (ii) β is a PRD $r = p[\vec{X}]$; or
- (iii) β is a DD $p = r_1 \mid \cdots \mid r_n$.

Let $\bar{\mathbf{D}}^\infty$ assign the decomposition $\bar{\pi}_r^\infty$ to r , $\bar{\pi}_p^\infty$ to p , etc. If β is an IND (case (i) above), consider some component $\bar{s}_r \in \bar{\pi}_r^\infty$. Suppose \bar{s}_r comes from a decomposition assigned to r by some intermediate database instance $\bar{\mathbf{D}}$. By the aforesaid properties of the yet-to-be-defined chase process, there is an

intermediate database instance, $\overline{\mathbf{D}}'$, that was constructed after $\overline{\mathbf{D}}$, such that β holds in $\overline{\mathbf{D}}'$. Hence, the decomposition that $\overline{\mathbf{D}}'$ assigns to p must have \overline{s}_r as a component. By construction of $\overline{\mathbf{D}}^\infty$, it follows that $\overline{s}_r \in \overline{\pi}_p^\infty$. Since \overline{s}_r is an arbitrary member of $\overline{\pi}_r^\infty$, we conclude that $\overline{\pi}_r^\infty \subseteq \overline{\pi}_p^\infty$, *i.e.*, $\overline{\mathbf{D}}^\infty$ satisfies β . Cases (ii) and (iii) are disposed of using similar arguments.

We have shown that $\overline{\mathbf{D}}^\infty$ satisfies Γ . But we started with $\overline{\mathbf{D}}^0$ that, by construction, violates α over the predicate q . Also, by construction of $\overline{\mathbf{D}}^\infty$, $\overline{\pi}_q^0 \subseteq \overline{\pi}_q^\infty$ (where $\overline{\pi}_q^0$ is the decomposition for q in $\overline{\mathbf{D}}^0$). Since α is violated in $\overline{\pi}_q^0$, it is violated in $\overline{\pi}_q^\infty$ hence also in $\overline{\mathbf{D}}^\infty$ — contrary to the assumption that $\Gamma \models \alpha$.

To complete the proof, it remains to do the following:

- to explain how compensating rules are applied in the chase process;
- to show that the intermediate database instances grow monotonically; and
- to prove that each rule-application transforms Γ -simple instances into Γ -simple ones.

Compensating rules come in three flavours: those triggered by the violation of IND's; those triggered by PRD's; and those triggered by DD's.

Suppose an IND $r \subseteq p$ is violated. Then $\overline{\pi}_r \not\subseteq \overline{\pi}_p$. Since $r : G(r)$ is obtained from Γ by exhaustive application of the inference rules, the IND-inference rule must have been applied for the IND $r \subseteq p$. Therefore, $r : G(p)$ must be derivable from $r : G(r)$. By the soundness of the inference rules,

$$r : G(r) \models r : G(p) \tag{3}$$

Thus, viewed as a decomposition over p , $\overline{\pi}_r$ is $G(p)$ -simple, by Corollary 7.5 (Claim 5). Application of the compensating rule that corresponds to the above IND consists of taking the union of $\overline{\pi}_r$ and $\overline{\pi}_p$ and making the result into the new decomposition $\overline{\pi}'_p$ over p (in the next intermediate database instance $\overline{\mathbf{D}}'$). Clearly, the IND is satisfied in the new database instance. By Claim 2 of Corollary 7.5, this union is also $G(p)$ -simple.

Suppose next that $r = p[\vec{X}]$ is a PRD violated by a Γ -simple database instance. We claim that

$$r : G(r) \models r : (G(p)[\vec{X}]) \quad \text{and} \quad r : (G(p)[\vec{X}]) \models r : G(r) \tag{4}$$

In proof, observe that $r : (G(p)[\vec{X}])$ is derived from $p : G(p)$, by the inference rules. By construction (and by soundness of the PRD-rules), $r : G(r)$ must logically entail every PC over r that is derivable from Γ by the inference rules. In particular, it must entail $r : (G(p)[\vec{X}])$. The second entailment follows because $p : \tau(G(r))$ must hold by rule PRD(ii), hence $G(p) \models \tau(G(r))$, by construction of $G(p)$. Then, applying Claim 3 of Corollary 7.3, we get $r : (G(p)[\vec{X}]) \models r : G(r)$.

Therefore, by Claim 5 of Corollary 7.5, any $G(r)$ -simple decomposition over r is also $G(p)[\vec{X}]$ -simple, and vice versa.

Now, if $\overline{\pi}_r \subseteq \overline{\pi}_p[\vec{X}]$ then we simply assign $\overline{\pi}_p[\vec{X}]$ to r . Since $\overline{\pi}_p$ is $G(p)$ -simple, $\overline{\pi}_p[\vec{X}]$ is $G(p)[\vec{X}]$ -simple (Claim 3 of Corollary 7.5) and, by our previous observation, it is $G(r)$ -simple.

In case $\bar{\pi}_r \not\subseteq \bar{\pi}_p[\vec{X}]$, note that $\bar{\pi}_r$ is $G(r)$ -simple. Thus, by the above observation, it is also $G(p)[\vec{X}]$ -simple. Hence, by Claim 4 of Corollary 7.5, we can take $\bar{\pi}_r$ and extend it to a $G(p)$ -simple decomposition on p . We then take a union of this extension with $\bar{\pi}_p$. The resulting decomposition, $\bar{\pi}'_p$, is also $G(p)$ -simple, by Claim 2 of Corollary 7.5. Therefore, we can make $\bar{\pi}'_p$ into the new decomposition over p . The new decomposition over r is then set to be the union of $\bar{\pi}_r$ and $\bar{\pi}_p[\vec{X}]$. This union is $G(r)$ -simple, because $\bar{\pi}_p[\vec{X}]$ is $G(p)[\vec{X}]$ -simple and, by (4), also $G(r)$ -simple. Clearly, this construction ensures that the PRD becomes satisfied in the new database instance.

Finally, suppose $p = r_1 \mid \cdots \mid r_n$ is a decomposition dependency violated by a Γ -simple instance. Due to the IND-rules,

$$p : G(p) \models p : (G(r_1) \mid \cdots \mid G(r_n)) \quad \text{and} \quad r_i : G(r_i) \models r_i : G(p) \quad (5)$$

for all $i = 1, \dots, n$. Therefore, every $G(p)$ -simple decomposition is $(G(r_1) \mid \cdots \mid G(r_n))$ -simple, which follows by the left side of (5) and by Claim 5 of Corollary 7.5. By the right-hand side of (5), every $G(r_i)$ -simple decomposition is also $G(p)$ -simple. Note that since p and all r_i have the same schema, any decomposition over p can be viewed as a decomposition over any of the r_i 's, and so it is $G(p)$ -simple and/or $G(r_i)$ -simple over p if and only if the same holds over r_i .

Since $\bar{\pi}_p$ is $(G(r_1) \mid \cdots \mid G(r_n))$ -simple, each component of $\bar{\pi}_p$ is simple with respect to some component of some $G(r_i)$. For each $G(r_i)$, let $\bar{\pi}_i$ be the decomposition whose components are precisely the components of $\bar{\pi}_p$ that are simple with respect to some member of $G(r_i)$ ($\bar{\pi}_i$ may turn out to be an empty set of relations). Then $\bar{\pi}_p = \bar{\pi}_1 \mid \cdots \mid \bar{\pi}_n$ and each $\bar{\pi}_i$ is $G(r_i)$ -simple. Now, we can apply our DD by taking the union of $\bar{\pi}_i$ and $\bar{\pi}_{r_i}$ (for each i) and making the resulting decomposition into the new instance $\bar{\pi}'_{r_i}$ over r_i . $G(r_i)$ -simplicity of $\bar{\pi}'_{r_i}$ follows from Corollary 7.5, Claim 2.

For the new decomposition $\bar{\pi}'_p$, we can take the union of $\bar{\pi}_p$ with all $\bar{\pi}_{r_i}$, $i = 1, \dots, n$. Since each $\bar{\pi}_{r_i}$ is $G(r_i)$ -simple, the observations made after (5) ensure that $\bar{\pi}_{r_i}$ is $G(p)$ -simple. Hence, by Claim 2 of Corollary 7.5, the union, $\bar{\pi}'_p$, is also $G(p)$ -simple. These two actions obviously make the resulting database instance satisfy the DD.

Clearly, each compensating rule has the effect of forcing satisfaction of the dependency that triggered the rule. It is also clear that the database instance grows monotonically in each case. Hence, the chase process has the properties we required for constructing the limit $\bar{\mathbf{D}}^\infty$, which completes the proof. \square

Having completed the axiomatization of our constraints, it is now easy to see that the membership problem for PC's (*i.e.*, the question of whether a PC is a logical consequence of a set of constraints) is decidable.

Theorem 9.2 *There is an algorithm that takes a PC α and a set Γ of PC's, IND's, DD's, and PRD's, and verifies whether $\Gamma \models \alpha$.*

Proof: The algorithm consists in applying the inference rules until no more rules can be applied. Since the number of different inequivalent sets of PC's is finite and since all inference rules generate new PC's (and FC's, as a special case), after a certain point no new PC will be produced. At this point, the algorithm terminates. The termination condition can be effectively checked, since equivalence of PC's is decidable, by Corollary 6.4. \square

The result of Theorem 9.2 is somewhat unexpected since the corresponding result for FD's and IND's is negative: the inference problem is neither axiomatizable (in the conventional sense) [5], nor decidable [24, 6]. Axiomatizability is particularly surprising because FC's cannot be expressed in first-order logic, while FD's are easily expressible as first-order formulas. Despite the many similarities between FC's and FD's there are some important differences. First, if all relations are finite then all FC's become trivial, and the problem reduces to that of inferring IND's only, which is decidable [5]. Second, one of the axioms for FD's and IND's is no longer sound for FC's.¹²

Corollary 9.3 *The time complexity of the algorithm in Theorem 9.2 has exponential upper bound in the size of Γ and α .*

Proof: Replacing a set of PC's with a single equivalent PC (and PC-inference over a single predicate in general) takes exponential number of steps. Projecting a PC also takes exponential time, as remarked earlier. The rules for applying DD's and IND's, on the other hand, require only a polynomial number of steps. Observe further that no inference rule needs to be applied more than an exponential number of times. This is because no rule needs to be applied twice with the same premises, and the number of all possible PC's is at most exponential. \square

In [8], it is shown that the corresponding inference problem for FD's and *acyclic* IND's is decidable, but is NP-hard. We do not know if the inference problem for FC's is also NP-hard. However, in the next section we show that the worst-case complexity of inferring *all* FC's that hold in the query predicate (in all fixpoints of the IDB) is exponential both in time and space.

10 Deciding Superfiniteness and Super-entailment

In this section we reduce the problems of superfiniteness and super-entailment for Horn queries to the problem of inferring PC's from constraints introduced in previous sections. Since the latter problem has been shown to be decidable in Theorem 9.2, so must be the other two problems.

Clearly, any sound super-entailment algorithm is also sound (but incomplete) for inferring FC's that hold in the *least* fixpoint. Moreover, we believe that, from the practical point of view, FC-inference is more important than query finiteness, since knowing the FC's may help to determine if a query evaluation process terminates [20, 7, 17], while mere knowledge that some query is finite is usually insufficient for detecting termination.

We remind that an FC α is *super-entailed* by an IDB \mathbf{P} and a set of FC's F if α holds in every fixpoint of \mathbf{P} that satisfies F . In this terminology, superfiniteness of q simply means super-entailment of an FC of the form $q : \text{Finite}(1, \dots, n)$, where n is the arity of q . Notice that here we are talking specifically about FC's, not just any PC. Although it still makes sense to talk about super-entailment of PC's, our decision procedure is correct for FC's only (but PC's are used in the process).

¹²The axiom in question is: **from** $(U \cup V) \subseteq (X \cup Y)$, $(U \cup W) \subseteq (X \cup Z)$ and $X \twoheadrightarrow Y$ **infer** $(U \cup V \cup W) \subseteq (X \cup Y \cup Z)$ [5, 24].

Reduction of Super-entailment and Superfiniteness to PC Inference

The reduction was already informally described in Examples 5.1, 5.2, and 7.1. To make this construction precise, we associate database schemas to Horn IDBs as follows. Given an IDB \mathbf{P} , the associated database schema \mathbf{D} consists of the following three groups of predicates:

- *Group 1*: All predicates mentioned in \mathbf{P} .
- *Group 2*: A new predicate *per each occurrence* of a literal in \mathbf{P} . Let literal $q(\vec{X})$ occur somewhere in \mathbf{P} . Then the new predicate in \mathbf{D} that is associated with this specific occurrence of $q(\vec{X})$ has the same arity as q .
- *Group 3*: A new predicate *per each rule* of \mathbf{P} . Associated with each Horn rule R is a predicate of arity equal the number of distinct variables in R . We shall use the symbol R both for the rule and for the corresponding predicate.

Apart from the predicates, \mathbf{D} has a set of constraints, denoted $\mathcal{C}(\mathbf{P})$. These constraints include the FC's that come originally with \mathbf{P} (usually they are obtained from the information about finiteness of the base predicates and via the process of function symbol elimination from [26], which is outlined in Section 4). Other constraints are derived from the structure of \mathbf{P} .

Formally, the constraints in $\mathcal{C}(\mathbf{P})$ come in three different flavors:

- *Category 1*: For each predicate symbol g in \mathbf{P} constrained by a set of FC's F (that come with \mathbf{P}), $\mathcal{C}(\mathbf{P})$ has a singleton PC of the form $g : (F)$.
- *Category 2*: For each occurrence of a literal $p(\vec{Z})$ in the *body* of a rule R , $\mathcal{C}(\mathbf{P})$ has:
 - a PRD of the form $p' = R[\vec{Z}]$;¹³ and
 - an IND $p \supseteq p'$.

Here p' denotes a Group 2 predicate symbol that \mathbf{D} associates with this particular *body*-occurrence of the literal $p(\vec{Z})$, and p is a Group 1 predicate of \mathbf{D} corresponding to the predicate p mentioned in \mathbf{P} . The above pair of dependencies says two things: 1) that the tuples used by R to instantiate the body literal $p(\vec{Z})$ must all come from the relation assigned to p ; *and* 2) that these tuples must match the projection pattern specified by the sequence of variables \vec{Z} .

- *Category 3*: For each IDB predicate p , $\mathcal{C}(\mathbf{P})$ has a DD and several PRD's constructed as follows. Let

$$\begin{array}{l} R^{(1)} : p(\vec{X}_1) \leftarrow \dots \\ \vdots \qquad \qquad \qquad \vdots \\ R^{(k)} : p(\vec{X}_k) \leftarrow \dots \end{array}$$

be all the rules in \mathbf{P} that have p in the *head*. Then $\mathcal{C}(\mathbf{P})$ includes the following constraints:

¹³We assume that when R is considered as a predicate, its attributes are named after the distinct variables of the rule R . So, here \vec{Z} is treated as a list of attributes of the rule-predicate R .

- k PRD's of the form $p^{(1)} = R^{(1)}[\vec{X}_1], \dots, p^{(k)} = R^{(k)}[\vec{X}_k]$; and
- a DD $p = p^{(1)} \mid \dots \mid p^{(k)}$.

Here each $R^{(i)}$ is a Group 3 predicate symbol that corresponds to the rule $R^{(i)}$; each $p^{(i)}$ is a Group 2 predicate symbol that \mathbf{D} associates with the *head*-occurrence of p in $R^{(i)}$; and p is a Group 1 predicate that comes from \mathbf{P} itself. The constraints in Category 3 express the fact that all p -tuples are generated through (and only through) the above k rules in \mathbf{P} .

The reduction of superfiniteness to the inference problem for PC's can now be stated as follows; its correctness is the subject of Theorem 10.2.

A Naive Decision Procedure for Super-entailment:

Let \mathbf{P} be a Horn IDB and F be a set of FC's. Construct the corresponding set of constraints $\mathcal{C}(\mathbf{P})$. Use the algorithm in Theorem 9.2 to decide whether an FC α can be derived by the inference rules. If it can, then α is super-entailed by \mathbf{P} and F ; otherwise, it is not super-entailed.

In view of the earlier discussions, superfiniteness of an n -ary predicate q can be decided by the above algorithm with α being $q : \text{Finite}(1, \dots, n)$.

Correctness of the Naive Decision Procedure

The central part in the proof of decidability of super-entailment is Lemma 10.1 below. It uses a construction that associates a special relation to each rule in \mathbf{P} . This construction was already informally described in Examples 5.1, 5.2, and 7.1. Given an interpretation, M , and a rule, R , with distinct variables V_1, \dots, V_n , the relation \bar{R}_M associated with R (or just \bar{R} , when M is known) has n attributes V_1, \dots, V_n named after the variables; it consists of all tuples $\langle a_1, \dots, a_n \rangle$, where the a_i 's are drawn from the domain of M ,¹⁴ such that after substituting a_i for V_i ($i = 1, \dots, n$) we get a ground instance of R whose body is true in M .

Lemma 10.1 *Let \mathbf{P} be an IDB constrained by a set of FC's and let \mathbf{D} be the database schema constructed earlier.*

1. *Let M be a fixpoint model of \mathbf{P} that satisfies the FC's that come with \mathbf{P} . Then there is an extended database instance $\bar{\mathbf{D}}^\infty$ over the schema \mathbf{D} such that $\bar{\mathbf{D}}^\infty$ satisfies $\mathcal{C}(\mathbf{P})$ and for every predicate symbol p in \mathbf{P} ,*

$$\bar{p}_M = \cup \bar{\pi}_p^\infty \tag{6}$$

where \bar{p}_M (resp., $\bar{\pi}_p^\infty$) is the relation (resp., decomposition of \bar{p}_M) that M (resp., $\bar{\mathbf{D}}^\infty$) assigns to p .¹⁵

¹⁴We assume some fixed order in which variables of R (and the attributes of \bar{R}_M) are listed.

¹⁵We remind that the notation $\cup \pi$ was introduced in Section 6 to denote a set-union of all components of the decomposition π .

2. Let $\overline{\mathbf{D}}$ be an extended database instance of \mathbf{D} that satisfies $\mathcal{C}(\mathbf{P})$. Then there is a fixpoint model M of \mathbf{P} such that

$$\overline{p}_M \supseteq \cup \overline{\pi}_p \quad (7)$$

where \overline{p}_M (resp., $\overline{\pi}_p$) is the relation (resp., decomposition) that M (resp., $\overline{\mathbf{D}}$) assigns to p .

Proof: Claim 1. Given M , we first construct $\overline{\mathbf{D}}^0$ — an initial database instance over schema \mathbf{D} . $\overline{\mathbf{D}}^0$ assigns an empty relation to every predicate of \mathbf{D} in Group 2 (which are predicates associated with literal-occurrences in \mathbf{P}). Predicates in Group 1 (i.e., the predicates mentioned in \mathbf{P}) are assigned the same relations as the relations assigned to them by M . Finally, for any Group 3 predicate R (i.e., one that is associated with a rule in \mathbf{P}), $\overline{\mathbf{D}}^0$ assigns \overline{R}_M —the relation described just before the statement of this lemma. We view $\overline{\mathbf{D}}^0$ as an extended database instance, where every decomposition is a singleton with just one component.

Because M is a model that satisfies the FC's that come with \mathbf{P} , $\overline{\mathbf{D}}^0$ satisfies all the PC's in $\mathcal{C}(\mathbf{P})$, since the lemma assumes that the only PC's in $\mathcal{C}(\mathbf{P})$ are FC's.¹⁶ However, some IND's, PRD's, or DD's of $\mathcal{C}(\mathbf{P})$ may be violated by $\overline{\mathbf{D}}^0$. To enforce these constraints, we transform $\overline{\mathbf{D}}^0$ in two stages. First, we apply a chase-like process to obtain a possibly infinite sequence of extended database instances that, informally speaking, are in “increasing compliance” with the constraints. Then we construct $\overline{\mathbf{D}}^\infty$ as a limit of sorts for this sequence.

The overall plan of the proof is similar to that of Proposition 9.1, but the chase process and the limit-construction are quite different. The construction in Proposition 9.1 cannot be used here because now the result of the chase must comply with Condition (6) above, so we can neither use $\mathcal{C}(\mathbf{P})$ -simple decompositions, nor we can freely add new components or tuples to the existing decompositions. In other words, we can no longer use Corollary 7.5, especially Claim 4 there.

Suppose p is a predicate in Group 1. It follows from Property (c), below, and from the yet to-be-described construction of the limit that $\overline{\pi}_p^\infty$ is, indeed, a decomposition of \overline{p}_M . Hence, Condition (6) above is satisfied and so are all the FC's in $\mathcal{C}(\mathbf{P})$. The latter is true since each \overline{p}_M is a finite approximation of $\overline{\pi}_p^\infty$, and \overline{p}_M satisfies the FC's in $\mathcal{C}(\mathbf{P})$, by the assumptions in the lemma. Thus, our main goal is to ensure that $\overline{\mathbf{D}}^\infty$ satisfies all the other types of constraints as well.

We will rely on the following properties that will be preserved throughout the chase process. Let $\overline{\mathbf{D}}^0, \overline{\mathbf{D}}^1, \dots$ be a sequence of database instances constructed by the chase process. For all j , let us assume that $\overline{\mathbf{D}}^j$ assigns a decomposition $\overline{\pi}_p^j$ to p , a decomposition $\overline{\pi}_{p^{(i)}}^j$ to $p^{(i)}$, etc. Then, this sequence of database instances has the following properties:

(a) Let $\overline{\pi}_q^j$ and $\overline{\pi}_q^{j+1}$ be decompositions that $\overline{\mathbf{D}}^j$ and $\overline{\mathbf{D}}^{j+1}$ assign to some arbitrary predicate q in \mathbf{D} . Let \overline{s}^j be some component of $\overline{\pi}_q^j$. Then

- either \overline{s}^j is also a component in $\overline{\pi}_q^{j+1}$;
- or $\overline{s}^j \notin \overline{\pi}_q^{j+1}$, $\overline{s}^j = \overline{s}_1^{j+1} \cup \dots \cup \overline{s}_n^{j+1}$, for some $n > 1$, and $\overline{s}_i^{j+1} \in \overline{\pi}_q^{j+1}$, $i = 1, \dots, n$.

This condition essentially says that components in our decompositions do not die out during the chase process. They are either passed along to a successor-decomposition (the first case)

¹⁶Since M is *not* an extended database instance, all PC's satisfied there must, in fact, be FC's.

or they are decomposed further (the second case). However, *new components* may appear in successor-decompositions—components that do not originate in preceding decompositions.

- (b) For each constraint (DD, PRD, or IND) in $\mathcal{C}(\mathbf{P})$ and for each $j \geq 0$, there is $k > j$ such that the constraint is satisfied in $\overline{\mathbf{D}}^k$. In particular, these constraints are satisfied infinitely often by the sequence $\overline{\mathbf{D}}^0, \overline{\mathbf{D}}^1, \dots$ (we view this sequence as if it were infinite; if it is finite, we assume that the last database instance is replicated infinitely many times).
- (c) The relations assigned to predicates in Groups 1 and 3 do not change. That is if p is a predicate that comes from \mathbf{P} then $\cup \overline{\pi}_p^j = \overline{p}_M$, for all j . Likewise, if $\overline{\pi}_R^j$ is a decomposition assigned to a rule-predicate R , then $\cup \overline{\pi}_R^j = \overline{R}_M$. Therefore, all FC's in $\mathcal{C}(\mathbf{P})$ are satisfied in all $\overline{\mathbf{D}}^j$, as \overline{p}_M is a finite approximation of $\overline{\pi}_p^j$, and \overline{p}_M satisfies the relevant FC's in $\mathcal{C}(\mathbf{P})$, by the assumption in the statement of the lemma.
- (d) The relations assigned to predicates in Group 2 grow monotonically, but they are bound by the relations assigned to predicates in Groups 1 and 3. More precisely, if p' is a Group 2 predicate corresponding to a literal-occurrence of a predicate p in \mathbf{P} that gives rise to a PRD $p' = R[\vec{X}]$ (which may be a constraint of Category 2 or 3, depending on whether the literal occurs in the body of R or in its head), then
- $\cup \overline{\pi}_{p'}^j \subseteq \cup \overline{\pi}_{p'}^{j+1}$,
 - $\cup \overline{\pi}_{p'}^j \subseteq \overline{R}_M[\vec{X}]$; and
 - $\cup \overline{\pi}_{p'}^j \subseteq \overline{p}_M$.

Note that (the applicable parts of) properties (a) through (d) are satisfied by the initial instance $\overline{\mathbf{D}}^0$.

The requisite extended database instance $\overline{\mathbf{D}}^\infty$ is a *limit* of the above sequence $\overline{\mathbf{D}}^0, \overline{\mathbf{D}}^1, \dots$; it is defined as follows. Suppose q is a predicate in \mathbf{D} and $\overline{\pi}_q^0, \overline{\pi}_q^1$, etc., are the decompositions assigned to q by the database instances in the sequence. The *limit* of this sequence, $\overline{\pi}_q^\infty$, is essentially an intersection of the decompositions in the sequence, with a small twist needed to accommodate the components of the decompositions that pop up during the chase process and which do not originate in earlier decompositions (as explained in Property (a)). The *limit instance* $\overline{\mathbf{D}}^\infty$ then assigns $\overline{\pi}_q^\infty$ to q , *i.e.*, it assigns an appropriate limit-decomposition to each predicate symbol in \mathbf{D} .

Formally, $\overline{\pi}_q^\infty$ is constructed as follows. By Property (a) above, each component of each $\overline{\pi}_q^j$ is part of a shrinking sequence of relations

$$\overline{s}_m \supseteq \overline{s}_{m+1} \supseteq \overline{s}_{m+2} \supseteq \dots \quad \text{for some } m \geq 0 \quad (8)$$

where, for each $j \geq m$, we have $\overline{s}_j \in \overline{\pi}_q^j$. Such a sequence may not always originate in $\overline{\pi}_q^0$, because, in the course of the chase process, decompositions may acquire new components, which do not originate in earlier decompositions (see Property (a)). The *limit* of $\overline{\pi}_q^0, \overline{\pi}_q^1, \dots$ is the following decomposition:

$$\overline{\pi}_q^\infty = \{ \cap_{j \geq m} \overline{s}_j \mid \overline{s}_m \supseteq \overline{s}_{m+1} \supseteq \overline{s}_{m+2} \supseteq \dots \text{ is a sequence such that } \overline{s}_j \in \overline{\pi}_q^j \text{ for each } j \geq m \}$$

Properties (a) and (c) ensure that $\overline{\pi}_q^\infty$ is a decomposition of \overline{q}_M , for each predicate q in \mathbf{D} that belongs to Group 1 or 3. We will now show that $\mathcal{C}(\mathbf{P})$ is satisfied in $\overline{\mathbf{D}}^\infty$.

The case of PC's has already been argued. Consider a PRD of the form $q' = q[\vec{Z}]$. If (8) is a sequence of relations that gives rise to a component of $\bar{\pi}_q^\infty$ of the form $\cap_{j \geq m} \bar{s}_j$, then, by Property (b), there is a sequence

$$\bar{s}'_{m'} \supseteq \bar{s}'_{m'+1} \supseteq \bar{s}'_{m'+2} \supseteq \dots \quad \text{for some } m' \geq 0$$

of components of $\bar{\pi}_{q'}^{m'}$, $\bar{\pi}_{q'}^{m'+1}$, ..., respectively, such that $\bar{s}'_j = \bar{s}_j[\vec{Z}]$ holds for infinitely many $j \geq \max(m, m')$.¹⁷ Therefore, $\cap_{j \geq m'} \bar{s}'_j = (\cap_{j \geq m} \bar{s}_j)[\vec{Z}]$, i. e., $\bar{\pi}_{q'}^\infty[\vec{Z}] \subseteq \bar{\pi}_q^\infty$. In the other direction, we can similarly show that every shrinking sequence over q' gives rise to a shrinking sequence over q , and that the corresponding PRD is satisfied in the limit. Hence, $\bar{\pi}_{q'}^\infty \subseteq \bar{\pi}_q^\infty[\vec{Z}]$ and the PRD $q' = q[\vec{Z}]$ is satisfied in $\bar{\mathbf{D}}^\infty$.

For IND's and DD's, the proof is similar. For instance, in case of a DD $p = p^{(1)} \mid \dots \mid p^{(k)}$, we have a sequence of decompositions $\bar{\pi}_p^0, \bar{\pi}_p^1, \dots$ over p and also decomposition sequences $\bar{\pi}_{p^{(i)}}^0, \bar{\pi}_{p^{(i)}}^1, \dots$ for each $p^{(i)}$. Again, by Property (b), the dependency

$$\bar{\pi}_p^j = \bar{\pi}_{p^{(1)}}^j \mid \dots \mid \bar{\pi}_{p^{(k)}}^j$$

holds for infinitely many j . As with PRD's, by considering sequences of the individual components of these decompositions, we can show that this dependency holds in the limit.

Having constructed the limit instance and shown that it satisfies all constraints in $\mathcal{C}(\mathbf{P})$, it remains to describe a chase process that can generate a sequence of extended database instances that satisfies properties (a) through (d).

Each chase action described below takes a current intermediate database instance $\bar{\mathbf{D}}^j$, identifies decompositions that violate some constraint, and then modifies some of the decompositions to resolve the problem. The successor instance $\bar{\mathbf{D}}^{j+1}$ is obtained by replacing the original decompositions with the modified ones; the rest of the decompositions of $\bar{\mathbf{D}}^j$ are passed along to $\bar{\mathbf{D}}^{j+1}$ without change. Since each chase action restores one violated constraint, Property (b) above will be satisfied, provided that the chase process picks constraints from $\mathcal{C}(\mathbf{P})$ in a fair manner. Therefore, we only need to show that Properties (a), (c), and (d) are satisfied by the sequence of instances constructed during the chase.

We start with a chase action aimed at restoring satisfaction of the DD's. Suppose that, say, $p = p^{(1)} \mid \dots \mid p^{(k)}$ is violated in a current intermediate instance $\bar{\mathbf{D}}^j$. This can happen in two ways.

In case $\bar{\pi}_{p^{(1)}}^j \mid \dots \mid \bar{\pi}_{p^{(k)}}^j \not\subseteq \bar{\pi}_p^j$, we construct $\bar{\pi}_p^{j+1}$ so that it would contain all the components of $\bar{\pi}_p^j$ plus all the offending components from each of the $\bar{\pi}_{p^{(i)}}^j$, $i = 1, \dots, k$.

Since $\bar{\mathbf{D}}^j$ satisfies Property (d), the newly added components of $\bar{\pi}_p^{j+1}$ must be subrelations of \bar{p}_M , so $\bar{\pi}_p^{j+1}$ remains a decomposition of \bar{p}_M and Property (c) is preserved. Obviously, the transition from $\bar{\pi}_p^j$ to $\bar{\pi}_p^{j+1}$ preserves Property (a). Property (d) holds since we did not touch decompositions of Group 2.

In the other direction, suppose $\bar{\pi}_p^j \not\subseteq \bar{\pi}_{p^{(1)}}^j \mid \dots \mid \bar{\pi}_{p^{(k)}}^j$. Then $\bar{\pi}_p^j$ has an offending component \bar{p}_o , which is not among the components of $\bar{\pi}_{p^{(1)}}^j, \dots, \bar{\pi}_{p^{(k)}}^j$. We can split \bar{p}_o into k components of the form

¹⁷Indeed, Property (b) states that $q' = q[\vec{Z}]$ is satisfied by infinitely many $\bar{\mathbf{D}}^j$, $j \geq \max(m, m')$, so the equation $\bar{\pi}_{q'}^j = \bar{\pi}_q^j[\vec{Z}]$ must hold for infinitely many j .

$\bar{p}_o \cap \bar{R}_M^{(i)}[\vec{X}]$, $i = 1, \dots, k$, and make these relations into components of $\bar{\pi}_p^{j+1}$ (here $R^{(i)}$ denotes the rule of \mathbf{P} that gives rise to $p^{(i)}$ —see the definition of constraints of Category 3). The component \bar{p}_o itself is *not* passed along to $\bar{\pi}_p^{j+1}$, but all the other members of $\bar{\pi}_p^j$ are. In addition, each $\bar{p}_o \cap \bar{R}_M^{(i)}[\vec{X}]$ becomes a component of $\bar{\pi}_{p^{(i)}}^{j+1}$ along with all the components of $\bar{\pi}_{p^{(i)}}^j$.

Note that the fact that M is a fixpoint model of \mathbf{P} is crucial in order for $\bar{\mathbf{D}}^{j+1}$ to satisfy properties (a) and (c). Indeed, because M is a fixpoint, $\bar{p}_o \subseteq \cup_{i=1}^k \bar{R}_M^{(i)}[\vec{X}]$ thus $\bar{\pi}_p^{j+1}$ remains a decomposition of \bar{p}_M . Verifying Property (d) is straightforward.

We now describe the chase action aimed at the restoration of an IND of the form $p \supseteq p'$. Notice that all IND's are Category 2 constraints, where p is always a predicate name in \mathbf{P} and p' is a Group 2 predicate that corresponds to a body-occurrence of p . Such an IND can be violated only if $\bar{\pi}_{p'}^j \not\subseteq \bar{\pi}_p^j$. Restoring IND-satisfaction is easy: just put all the offending components of $\bar{\pi}_{p'}^j$ into $\bar{\pi}_p^{j+1}$ and also copy the components of $\bar{\pi}_p^j$ there. Property (d) of $\bar{\mathbf{D}}^j$ guarantees that Property (c) holds in $\bar{\mathbf{D}}^{j+1}$. Property (a) is satisfied trivially and Property (d) continues to hold since the construction of $\bar{\pi}_p^{j+1}$ adds more components to Group 1 decompositions, but does not affect decompositions of predicates in Group 2.

The chase action for the PRD's is defined as follows. Suppose $p' = R[\vec{Z}]$ is violated for some rule R (this also applies to $p^{(i)} = R[\vec{X}]$, a PRD of Category 3). If $\bar{r}' \in \bar{\pi}_{p'}^j$ is an offending component, *i.e.*, there is no component in $\bar{\pi}_R^j$ that projects onto \bar{r}' , then we choose a subrelation $\bar{r} \subseteq \bar{R}_M$, such that $\bar{r}[\vec{Z}] = \bar{r}'$ and make \bar{r} into a component of $\bar{\pi}_R^{j+1}$. The relation \bar{r} can always be found because Property (d) ensures that $\cup \bar{\pi}_{p'}^j \subseteq \bar{R}_M[\vec{Z}]$.¹⁸ In addition, we copy all the components of $\bar{\pi}_R^j$ into $\bar{\pi}_R^{j+1}$. Again, properties (a), (c), and (d) are satisfied by $\bar{\mathbf{D}}^{j+1}$.

In the other direction, if $\bar{\pi}_R^j$ has an offending component, \bar{r} , such that $\bar{r}[\vec{Z}]$ is not in $\bar{\pi}_{p'}^j$, then we simply add $\bar{r}[\vec{Z}]$ to $\bar{\pi}_{p'}^{j+1}$ along with all the components of $\bar{\pi}_{p'}^j$. Property (d) is satisfied by $\bar{\mathbf{D}}^{j+1}$ because: 1) $\cup \bar{\pi}_{p'}^j \subseteq \cup \bar{\pi}_{p'}^{j+1}$, by construction; 2) $\cup \bar{\pi}_{p'}^{j+1} \subseteq \bar{R}_M[\vec{Z}]$, since $\cup \bar{\pi}_{p'}^j \subseteq \bar{R}_M[\vec{Z}]$ (Property (d) for $\bar{\mathbf{D}}^j$) and because $\bar{r}[\vec{Z}] \subseteq \bar{R}_M[\vec{Z}]$; and 3) $\cup \bar{\pi}_{p'}^{j+1} \subseteq \bar{p}_M$, due to 2) and the fact that $\bar{R}_M[\vec{Z}] \subseteq \bar{p}_M$ (which holds since M is a model of \mathbf{P}).

Claim 2. Construct a model using $\bar{\mathbf{D}}$ as follows. Let p be a predicate symbol in \mathbf{P} , and let $\bar{\pi}_p$ be the decomposition assigned to it by $\bar{\mathbf{D}}$. Let I be the interpretation of \mathbf{P} that assigns to each predicate p in \mathbf{P} the relation $\bar{p} = \cup \bar{\pi}_p$. The resulting interpretation might not be a model, though. Indeed, I might contain atomic facts that match the body of a rule in \mathbf{P} , but it might not contain the appropriate fact to satisfy the head of the rule.

To obtain the requisite model, we simply apply the rules of \mathbf{P} to I in a bottom-up manner and continue until no new facts can be generated. The result, M , is a model of \mathbf{P} that contains I : it is a model because it was obtained via a bottom-up computation applied to a set of facts, and it contains I because I was that initial set of facts.

¹⁸Note that this simple trick was not possible in Proposition 9.1 where we also dealt with chasing PRD's. This is because, in that proposition, such operation would not guarantee that all PC's would remain satisfied. This problem was solved there via the use of Claim 4 of Corollary 7.5.

The model M satisfies all the requirements of the lemma: it is a fixpoint model, as we shall show shortly, and Condition (7) holds, by construction.

To show that M is a fixpoint, we need to establish that $T_{\mathbf{P} \cup \mathbf{edb}}(M) \subseteq M$ and $M \subseteq T_{\mathbf{P} \cup \mathbf{edb}}(M)$. The former is just a re-statement of the fact that M is a model of $\mathbf{P} \cup \mathbf{edb}$ [22], as mentioned in Section 2. The latter property is called *supportedness*; it means that every fact in M is either an \mathbf{edb} -fact, or it can be derived with an appropriate rule of \mathbf{P} applied to the appropriate facts of M .

Recall that M is obtained from I through a bottom-up computation that exhaustively applies the rules of \mathbf{P} . So, if we show that every IDB fact of I is supported (*i.e.*, it can be obtained by applying a rule of \mathbf{P} to some facts from I), then supportedness of M will be established, since all the facts in $M - I$ were derived by the bottom-up computation, hence they are supported by the definition of that computation.

It thus remains to establish that I is a supported interpretation. Consider an arbitrary fact $p(t) \in I$. Let $R^{(1)}, \dots, R^{(k)}$ be all the rules in \mathbf{P} that have p as their head predicate. By construction of $\mathcal{C}(\mathbf{P})$, it has the DD $p = p^{(1)} \mid \dots \mid p^{(k)}$.

Let $\bar{\pi}_p, \bar{\pi}_{p^{(1)}}, \dots$, be the decompositions that $\bar{\mathbf{D}}$ assigns to the predicates $p, p^{(1)}, \dots$, respectively. Since $\bar{\mathbf{D}}$ satisfies $\mathcal{C}(\mathbf{P})$, we have $\bar{\pi}_p = \bar{\pi}_{p^{(1)}} \mid \dots \mid \bar{\pi}_{p^{(k)}}$. Therefore,

$$\bar{p} = \cup \bar{\pi}_p = \cup_{i=1}^k (\cup \bar{\pi}_{p^{(i)}})$$

where $\bar{p}, \bar{p}^{(1)}, \dots$, are the relations that I assigns to $p, p^{(1)}, \dots$, respectively. Note that since our chosen fact $p(t)$ is in I , it follows that $t \in \bar{p}$. Therefore, t must come from one of the $\bar{\pi}_{p^{(i)}}$; say $t \in \cup \bar{\pi}_{p^{(1)}}$, for definiteness. Suppose the rule $R^{(1)}$ has the form

$$p(\vec{X}) \leftarrow q_1(\vec{Z}_1), \dots, q_n(\vec{Z}_n)$$

Since $\mathcal{C}(\mathbf{P})$ contains the Category 3 PRD $p^{(1)} = R^{(1)}[\vec{X}]$, we have $\bar{\pi}_{p^{(1)}} = \bar{\pi}_{R^{(1)}}[\vec{X}]$. Together with $t \in \cup \bar{\pi}_{p^{(1)}}$, this implies that $t = t^{(1)}[\vec{X}]$, for some $t^{(1)} \in \cup \bar{\pi}_{R^{(1)}}$.

Consider now the relation $\bar{R}^{(1)}$ ($= \cup \bar{\pi}_{R^{(1)}}$), which I assigns to $R^{(1)}$ (the predicate name that schema \mathbf{D} associates with rule $R^{(1)}$). Since for each body literal q_j in $R^{(1)}$, $j = 1, \dots, n$, the set of dependencies $\mathcal{C}(\mathbf{P})$ includes the Category 2 dependencies $q'_j = R^{(1)}[\vec{Z}_j]$ and $q'_j \subseteq q_j$, for $j = 1, \dots, n$, it follows that $t^{(1)}[\vec{Z}_j] \in \cup \bar{\pi}_{q_j} = \bar{q}_j$, where $\bar{\pi}_{q_j}$ is the decomposition that $\bar{\mathbf{D}}$ assigns to q_j , and \bar{q}_j is the relation that I assigns to q_j . But this means precisely that $p(t)$ is derivable via rule $R^{(1)}$ when it is applied to the tuples $t^{(1)}[\vec{Z}_1] \in \bar{q}_1, \dots, t^{(1)}[\vec{Z}_n] \in \bar{q}_n$. Therefore, $p(t)$ is a supported fact in I . \square

Note that, while constructing M in the above proof, we allowed IDB-predicates to have some initial value. Therefore, M may not be the *least* model generated by applying $T_{\mathbf{P} \cup \mathbf{edb}}$, where \mathbf{edb} is the EDB-part of $\bar{\mathbf{D}}$. This explains why our method does not capture finiteness in the least fixpoint.

Theorem 10.2 *The problem of superfiniteness for Horn queries with FC's is decidable.*

Proof: We show that the naive decision procedure for superfiniteness (introduced in the first subsection of this section) is correct. Let \mathbf{P} be an IDB and q be an m -ary query predicate.

Soundness of the naive procedure follows from Lemma 10.1. Indeed, suppose that the algorithm says that $\mathcal{C}(\mathbf{P}) \models q : \text{Finite}(1, \dots, m)$ while, in fact, the query is not superfinite. Then \mathbf{P} has a

fixpoint model M that satisfies the given FC's and in which q is assigned an infinite relation \bar{q} . By Claim 1 of Lemma 10.1, there is an extended database instance that satisfies $\mathcal{C}(\mathbf{P})$ and assigns to q some decomposition of \bar{q} . But this contradicts soundness of the inference rules (Proposition 9.1).

To establish completeness of the naive procedure, suppose that the algorithm of Theorem 9.2 says that $\mathcal{C}(\mathbf{P}) \not\models q : \text{Finite}(1, \dots, m)$. Then, by completeness of the inference rules (Proposition 9.1), there is an extended database instance that satisfies $\mathcal{C}(\mathbf{P})$ and such that q is assigned a decomposition $\bar{\pi}_q$ with an infinite number of tuples. By Claim 2 of Lemma 10.1, there should be a fixpoint model, M , of \mathbf{P} that satisfies the FC's and such that $\bar{q} \supseteq \cup \bar{\pi}_q$. But this means that M assigns an infinite relation to q . \square

A similar result holds for super-entailment. Instead of proving it here, we establish a stronger result in the next subsection.

A Semi-naive Decision Procedure

We shall now present a *semi-naive* decision procedure for detecting superfiniteness and super-entailment. As stated earlier, superfiniteness is a special case of super-entailment, so this leads to a decision procedure for superfiniteness as well. The semi-naive procedure is more efficient than the naive algorithm introduced earlier, because it bypasses the application of certain inference rules.¹⁹ Another advantage of the semi-naive procedure is that it is more suitable for human use and comprehension. However, we do not call this procedure “semi-naive” for nothing—it retains the bottom-up derive-all naïveté of the old algorithm.

Our inference algorithm uses two basic operations: *induce* and *produce*. Consider a rule R of the form $r(\vec{X}) \leftarrow p_1(\vec{Z}_1), \dots, p_n(\vec{Z}_n)$ and let $R(V_1, \dots, V_k)$ be a Group 3 predicate for that rule. As before, V_1, \dots, V_n is a list of all distinct variables in the rule. The constraints in $\mathcal{C}(\mathbf{P})$ imply $R[\vec{Z}_i] \subseteq p_i$, for each i . Here we have used a *hybrid IND* $R[\vec{Z}_i] \subseteq p_i$, for better readability. A hybrid IND is an obvious combination of a PRD and an IND. The IND inference rule combined with the second PRD-rule implies that any PC on p_i induces some PC on R .

For easy reference, we spell out this inducement operation using the position-number notation. Suppose, in terms of the variables V_1, \dots, V_k , the variable list \vec{Z}_i can be written as V_{j_1}, \dots, V_{j_m} . Then we can re-write the above hybrid IND as $R[j_1, \dots, j_m] \subseteq p_i$. In Section 7, we introduced attribute mappings associated with PRD's along with their related promotion operations. These notions equally apply to hybrid IND's. In our concrete case, the attribute mapping associated with the above IND is: $\tau(p_i : l) = R : j_l$, $l = 1, \dots, m$, and the promotion of $\alpha = p_i : X \dashrightarrow Y$ is $\tau(\alpha) = R : \tau(X) \dashrightarrow \tau(Y)$. Promotion for sets of FC's and PC's with respect to hybrid IND's is defined exactly as for PRD's. The IND and the PRD(ii) inference rules then ensure that if a PC α holds over p_i then $\tau(\alpha)$ holds over R .

Let Γ be the set of PC's *induced* on R by all of its body literals. Then, we can compute the set of all PC's that hold over the variables in the head of the rule by projecting Γ on \vec{X} . We say that $\Gamma[\vec{X}]$

¹⁹In general, these inference rules *are* needed for completeness of PC-inference. They can be avoided in our semi-naive procedure because here we are dealing with $\mathcal{C}(\mathbf{P})$, a specialized set of constraints derived from \mathbf{P} .

is the set of PC's *produced* for the head predicate r . If \bar{r} is a relation computed for r , the above set of PC's may not hold in the whole of \bar{r} . However, it does hold in the part of \bar{r} generated by the rule R .

The complexity of producing PC's for the head predicate may be exponential in k —the number of distinct variables in R —as shown in [12]. In fact, Fischer et al. [12] have shown that for certain sets of FC's, the size of $F[\vec{X}]$ may be exponential in the size of F . However, Gottlob [14] later proposed an efficient algorithm that runs in polynomial time in many practical cases.²⁰

Algorithm 10.3 *Semi-naive Inference of FC's over IDB Predicates*

Input: *IDB \mathbf{P} and a set F of FC's for the EDB-predicates.*

As before, we shall use $\mathcal{C}(\mathbf{P})$ to denote the set of constraints initially derived from the structure of \mathbf{P} .

Output: *A PC for each IDB-predicate.*

Initialization:

For each predicate name p in \mathbf{P} , the algorithm uses $\text{pc}(p)$ to denote the current status of the PC computed for that predicate. For convenience, we represent $\text{pc}(p)$ as a set, although, as we already know, any set of PC's can be reduced to a single PC.

Initially, $\text{pc}(p)$ is some trivial PC, if p is an IDB-predicate; if p is an EDB-predicate then $\text{pc}(p)$ is the PC for this predicate that is given as input (as part of F).

Method: Repeat Steps 1 – 3 until no changes to any of the $\text{pc}(r)$ result:

1. For each rule, $R \in \mathbf{P}$, **induce** the PC's computed for the body literals onto the rule predicate.
2. For each rule $R \in \mathbf{P}$, **produce** the PC's for the head literal of R .
3. Let r be a head predicate defined by the rules $R^{(1)}, \dots, R^{(l)}$ and let $\text{pc}(r, R^{(i)})$ denote the PC produced for r by rule $R^{(i)}$ at Step 2. Then

Construct: $\text{pc}(r) := \{\text{pc}(r, R^{(1)}) \mid \dots \mid \text{pc}(r, R^{(l)})\} \cup \text{pc}(r)$ for every head predicate r in \mathbf{P} .

It is easy to see that the semi-naive algorithm terminates for all inputs. Indeed, after each iteration, the PC's $\text{pc}(r)$ are at least as strong as they were before the iteration (*i.e.*, they imply the old ones). Since there is only a finite number of possible PC's, their strength cannot grow indefinitely. At some point, the new iteration will leave all $\text{pc}(r)$'s unchanged, thereby terminating the algorithm.

Theorem 10.4 *The semi-naive algorithm leads to the following decision procedure for determining whether an FC, $r : \alpha$, is super-entailed by an IDB \mathbf{P} and a set of FC's, F :*

Compute $\text{pc}(r)$ using Algorithm 10.3. If $\text{pc}(r) \vdash \alpha$ (using the inference rules for PC's only) then \mathbf{P} and F super-entail $r : \alpha$. Otherwise, $r : \alpha$ is not super-entailed.

²⁰In [12] and [14], the results were actually obtained for FD's. However, they carry over to FC's, due to the fact that FC's and FD's have the same axiomatization when they are considered over a single predicate.

Proof: Let \mathbf{D} be the database schema constructed from \mathbf{P} (at the beginning of Section 10) and let $\mathcal{C}(\mathbf{P})$ be the set of constraints constructed from \mathbf{P} and F , as described earlier in this section.

First, observe that

$$\mathcal{C}(\mathbf{P}) \models r : \alpha \text{ if and only if } \alpha \text{ is derivable from } \mathbf{pc}(r) \text{ by PC-rules alone.} \quad (9)$$

where $\mathbf{pc}(r)$ is the PC computed for r by the semi-naive algorithm above. To see this, recall that our inference rules are complete for PC-inference over extended database instances. Therefore, our claim would follow from Proposition 9.1, if we prove that the semi-naive algorithm applies inference rules in all possible ways, except for some rules that can be shown to not advance the overall cause.

To find these “unproductive” inference rules, we first re-write the non-PC constraints in $\mathcal{C}(\mathbf{P})$ in the following form (where we will slightly abuse the notation by combining PRD’s with IND’s and DD’s):

- $R[\vec{Z}] \subseteq p$ — a combination of an IND and a PRD, where both belong to Category 2 of constraints (in our earlier classification); or
- $r = R^{(1)}[\vec{X}_1] \mid \cdots \mid R^{(l)}[\vec{X}_l]$ — a combination of a DD with l PRD’s, all belonging to Category 3 of constraints.

In Category 2, p may be an EDB or an IDB-predicate; the variable list \vec{Z} corresponds to the occurrence of $p(\vec{Z})$ in the body of R . In Category 3, r must be an IDB predicate and \vec{X}_i comes from the head occurrence of r in the rule $R^{(i)}$.

Initially, $\mathbf{pc}(r)$ is trivial for every IDB-predicate. Therefore, only the IND and PRD-rules corresponding to constraints in Category 2 need to be applied. This would be the first *induce* step. Notice that only the rule PRD(ii) is used here. Indeed, suppose we use $R[\vec{Z}] = p'$ to derive $\alpha[\vec{Z}]$ for the intermediate predicate p' (which would be a Group 2 predicate, according to our earlier classification of predicates in \mathbf{D}). Since p' corresponds to a body occurrence of p , the only other constraint it occurs in is $p' \subseteq p$. Clearly, $\alpha[\vec{Z}]$ cannot be used to derive new PC’s on p , for there is no IND-rule for doing this.

Following the *induce*-step, a *produce*-step (followed by a *construct*-step) would derive the PC $r : (\mathbf{pc}(r, R^{(1)}) \mid \cdots \mid \mathbf{pc}(r, R^{(l)}))$. At this stage, we can either apply inference rules corresponding to the constraints in Category 2 (another *induce*-step), or we can try the rules for the constraints in Category 3. The latter, however, are useless. Indeed, we can use them only to infer $\beta = \mathbf{pc}(r, R^{(1)}) \mid \cdots \mid \mathbf{pc}(r, R^{(l)})$ on each of the $r^{(i)}$ ’s, where $r^{(i)}$ is the Group 2 predicate in \mathbf{D} corresponding to the occurrence of r in the head of $R^{(i)}$. (By construction of $\mathcal{C}(\mathbf{P})$, $r^{(i)}$ occurs in the following two constraints: $R^{(i)}[\vec{X}] = r^{(i)}$ and $r = r^{(1)} \mid \cdots \mid r^{(l)}$.) But deriving β for $r^{(i)}$ would be a waste, since we have previously derived $\mathbf{pc}(r, R^{(i)})$ — a stronger PC over $r^{(i)}$.

These arguments, applied inductively, show that the rules PRD(i) and DD(ii) need never be applied because of the special structure of $\mathcal{C}(\mathbf{P})$. Since our algorithm applies all the other inference rules exhaustively, Claim (9) follows.

The rest of the proof uses Lemma 10.1 the same way as in Theorem 10.2. That is, suppose that some FC $p : \alpha$ does not hold in a fixpoint model of \mathbf{P} . Then we can construct an extended database

instance $\overline{\mathbf{D}}$ that, by Lemma 10.1, satisfies $\mathcal{C}(\mathbf{P})$ and Condition (6) (in the statement of that lemma). But this would then mean that α is violated by $\overline{\pi}_p$, the decomposition that $\overline{\mathbf{D}}$ assigns to p . Hence, $\mathcal{C}(\mathbf{P}) \not\models p : \alpha$ and our semi-naive algorithm will not derive this FC (by Claim (9) above).

For the other direction, suppose that our algorithm does not derive $p : \alpha$. Then, by Claim (9), $\mathcal{C}(\mathbf{P}) \not\models p : \alpha$ and there is an extended database instance $\overline{\mathbf{D}}$ that satisfies $\mathcal{C}(\mathbf{P})$ but violates $p : \alpha$. By Lemma 10.1, there is a fixpoint model of \mathbf{P} that satisfies Condition (7). Clearly, $p : \alpha$ is violated in that model as well. \square

It may be useful to note that the construction process for $\mathcal{C}(\mathbf{P})$ in Algorithm 10.3 and Theorems 10.2 and 10.4 does not depend on the assumption that the input set of dependencies is limited to FC's. In fact, all our arguments and constructions would go through even if the input contained PC's. However, the last part of the proof in Theorem 10.4 *does* rely on the assumption that the PC α there is, in fact, an FC.

As remarked earlier, the worst-case complexity of the above algorithm is exponential in the size of the largest rule in the IDB. However, this happens not due to some deficiency of our algorithm, but rather due to the exponential worst-case complexity of the problem at hand, both in time and space. This follows from the fact that in the realm of FD's (and FC's) over a single relation²¹ the size of the set of projected FD's may be exponential in the input [12]. Nevertheless, the results in [14] indicate that this happens only in pathological cases and that the use of the projection algorithm in [14] could make our semi-naive algorithm quite practical.

We do not know if there is a substantially more efficient way to determine whether a *given* FC holds in an IDB-predicate. There is a linear procedure for testing this in case of a single predicate [3], but it is unclear how this procedure can be used to help optimize FC-inference over IDB-predicates.

Examples

As a first application of the semi-naive algorithm, we shall prove superfiniteness of the IDB in Example 5.2. After the first induce-produce-construct sequence of steps, the algorithm will derive

$$p : (\mathbf{Finite}(1,2) \mid \mathbf{Finite}(2) \mid \mathbf{Finite}(1)) \tag{10}$$

The first component in (10) can be dropped, as this PC is equivalent to

$$p : (\mathbf{Finite}(2) \mid \mathbf{Finite}(1))$$

Table 1 details the FC's derived for the rule predicates R_1 , R_2 , and R_3 (of Example 5.2) at the "induce" stage of the algorithm. It also shows the FC's "produced" for the head predicate p by each rule; the PC (10) is constructed out of the latter FC's.

In the second iteration, additional PC's are induced, as depicted in Table 2. Applying the PC-inference rules to the PC's induced in the second stage (which are shown in the "induced" columns of Tables 1 and 2), we can derive: $R_2 : (\mathbf{Finite}(X_2) \mid \mathbf{Finite}(X_2))$ (which is $R_2 : \mathbf{Finite}(X_2)$) and $R_3 : (\mathbf{Finite}(Y_3) \mid \mathbf{Finite}(Y_3))$ (which is $R_3 : \mathbf{Finite}(Y_3)$). Since we already have the FC's

²¹And also in the realm of FC's over Group 3 predicates associated with Horn rules, as defined at the beginning of this section.

Rule	PC's Induced on Rule Predicates	PC's Produced for Rule Heads
R_1	$\text{Finite}(X_1)$	$p : \text{Finite}(1, 2)$
R_2	$V_2 \dashrightarrow X_2; W_2 \dashrightarrow X_2; \text{Finite}(Y_2)$	$p : \text{Finite}(2)$
R_3	$V_3 \dashrightarrow Y_3; W_3 \dashrightarrow Y_3; \text{Finite}(X_3)$	$p : \text{Finite}(1)$

Table 1: First Iteration: Induce and Produce Steps

Rule	PC's Induced on Rule Predicates	PC's Produced for Rule Heads
R_1	$\text{Finite}(X_1)$	$p : \text{Finite}(1, 2)$
R_2	$\text{Finite}(W_2) \mid \text{Finite}(V_2)$	$p : \text{Finite}(2); p : \text{Finite}(1)$
R_3	$\text{Finite}(W_3) \mid \text{Finite}(V_3)$	$p : \text{Finite}(1); p : \text{Finite}(2)$

Table 2: Second Iteration: Induce and Produce Steps

$R_2 : \text{Finite}(Y_2)$ and $R_3 : \text{Finite}(X_3)$, the “produce” stage yields the finiteness constraints depicted in Table 2. The “construct” step of the iteration then derives:

$$p : (\text{Finite}(1, 2) \mid \{\text{Finite}(1), \text{Finite}(2)\} \mid \{\text{Finite}(2), \text{Finite}(1)\}) \quad (11)$$

which is equivalent to $\text{Finite}(1, 2)$. Subsequent iterations do not bring new changes, and the algorithm terminates.

Example 10.5 (Another non-trivial, superfinite example) Let the IDB be:

$$\begin{aligned} R_1 &: p(X_1, Y_1) \leftarrow d(X_1), d(Y_1) \\ R_2 &: p(X_2, Y_2) \leftarrow f(X_2, Y_2), p(Y_2, Z_2), d(Z_2) \\ d &: \text{Finite}(1) \\ f &: 2 \dashrightarrow 1 \end{aligned}$$

The first iteration of the semi-naive algorithm yields $p : (\text{Finite}(1, 2) \mid 2 \dashrightarrow 1)$ and the second iteration derives $p : (\text{Finite}(1, 2) \mid \{\text{Finite}(1), \text{Finite}(2)\})$, which proves superfiniteness.

Note that without the literal $d(Z_2)$ in the second rule, the query is not finite and our algorithm would only derive the FC $p : 2 \dashrightarrow 1$. \square

Even though the above examples prove that superfiniteness is a useful notion, it is, unfortunately, a rather brittle one. An equivalence transformation may turn a superfinite query into a non-superfinite one. (Of course, here we are talking about equivalence with respect to the least fixpoint of the database; superfiniteness is obviously preserved under *uniform equivalence* introduced by Sagiv [29].)

Example 10.6 (Superfiniteness and query equivalence) Clearly, predicate p in the following IDB is superfinite:

$$\begin{aligned} p(X) &\leftarrow d(X) \\ d &: \text{Finite}(1) \end{aligned}$$

However, the addition of a seemingly innocuous rule, $p(Y) \leftarrow p(Y)$, turns p into a finite, but not a superfinite predicate. This brittleness is not that surprising, if we recall the well-known fact that Clark’s completion of any logic program breaks down under a similar transformation [22]. After all, superfiniteness means finiteness in all models of the Clark’s completion of the program.

It is easy to see that our algorithm stumbles on the above IDB (augmented with $p(Y) \leftarrow p(Y)$) right in the first iteration, where it produces the trivial FC, $p : (\text{Finite}(1) \mid \{\})$. \square

The next example presents a finite query that is non-superfinite for a much more subtle reason than in the previous example: the semi-naive algorithm cannot derive FC’s that would be sufficiently strong for proving query finiteness.

Example 10.7 (Finite, yet non-superfinite query) Consider the following IDB:

$$\begin{aligned} p(X, Y) &\leftarrow g(X, Y) \\ p(X, Y) &\leftarrow b(X, Z), p(Z, Y) \\ q(Y) &\leftarrow d(X), p(X, Y) \\ b &: \text{Finite}(1, 2) \\ d &: \text{Finite}(1) \\ g &: 1 \dashrightarrow 2 \end{aligned}$$

The extension of q is finite in the least fixpoint of this IDB (with an appropriate EDB). This is because it can be shown that $p : 1 \dashrightarrow 2$ holds in the least fixpoint. Predicate q is not superfinite because if b contains a “cyclic” tuple, say $\langle 0, 0 \rangle$, then it is easy to construct a fixpoint model where $p : 1 \rightarrow 2$ fails. For instance, the interpretation that assigns g and d the empty relations and b and p the relations $\{\langle 0, 0 \rangle\}$ and $\{\langle 0, n \rangle \mid n = 0, 1, 2, \dots\}$, respectively, is a fixpoint model of the above IDB where all the FC’s that are part of that IDB hold. Yet $p : 1 \dashrightarrow 2$ fails in this model.

Accordingly, our semi-naive inference algorithm only infers $p : (1 \dashrightarrow 2 \mid \text{Finite}(1))$ —not enough for proving $p : 1 \dashrightarrow 2$. Therefore, only the trivial FC can be derived for q . \square

11 Related Work and Open Problems

Various notions of finiteness have been studied for Datalog with function symbols and for Extended Datalog with different kinds of integrity constraints. In this section we classify the known results and mention some open problems.

We have shown that a notion stronger than finiteness, namely superfiniteness, is decidable for Extended Datalog with FC’s. Sagiv and Vardi proved that a weaker notion, *weak finiteness*, is decidable for Extended Datalog both with FC’s and FD’s [32].

As for the usual notion of finiteness, there are decidability results for special classes of IDBs. Consider Extended Datalog. If only constraints of the form $\text{Finite}(X)$ are allowed, finiteness is decidable [16, 17]. If we allow FC’s, the problem is known to be decidable (in polynomial time, in fact) for monadic IDBs [32]. The finiteness problem is also decidable for the following *non-recursive* IDBs:

1. *Extended Datalog with FC's*. This follows either from the results of this paper or from [32], since for non-recursive IDBs superfiniteness and weak finiteness coincide with finiteness.
2. *Extended Datalog with FD's (as opposed to FC's)*. This follows from [32], since weak finiteness is the same as finiteness for non-recursive IDBs.
3. *Datalog with function symbols [17]*.

On the other hand, Shmueli [36] has shown that finiteness is undecidable for (recursive) Datalog with function symbols. Sagiv and Vardi [32] proved that finiteness is undecidable for Extended Datalog with FD's, even for monadic IDBs. The following proposition shows that superfiniteness is also undecidable for Datalog with function symbols.

Proposition 11.1 *Superfiniteness of a query in Datalog with function symbols is recursively unsolvable.*

Proof: It is shown in [35, Theorem 13] that there exists a Horn IDB \mathbf{P} such that the set S of all negative ground literals that are true in all Herbrand models of $\text{comp}(\mathbf{P})$ (the Clark's completion of \mathbf{P} [22]), is not recursively enumerable. We will show that if superfiniteness for Datalog with function symbols were decidable, there would then be an algorithm to enumerate S .

Consider the IDB \mathbf{P} mentioned above, and let l be a ground atom taken from the Herbrand base of \mathbf{P} . Let us add the rule $g(X) \leftarrow l, g(f(X))$, where g is a new predicate symbol and f is a function symbol. Let us call the resulting IDB \mathbf{P}' . We claim that

$$?-g(X) \text{ is superfinite if and only if for every fixpoint model } M \text{ of } \mathbf{P}, M \models l \quad (12)$$

Indeed, if for some fixpoint model M of \mathbf{P} it were the case that $M \models l$, then we could extend M to a fixpoint model M' of \mathbf{P}' , where g would be infinite (thereby demonstrating that $?-g(X)$ is not a superfinite query). To see this, first add some $g(a)$ to M , where a is a constant. Then make sure that the fact $g(a)$ is supported in M' by adding the literals $g(f(a))$, $g(f(f(a)))$, etc. The resulting interpretation M' is a model of \mathbf{P} (since g is a new predicate, which does not occur in \mathbf{P}), and it is a fixpoint model of \mathbf{P}' , since we have saturated M' with the literals $g(f^n(a))$ that ensure that the newly added rule is satisfied and all the literals $g(f^n(a))$ are supported.

Conversely, if $?-g(X)$ is not superfinite, then g is infinite in some fixpoint model M' of \mathbf{P}' . Therefore, if some $g(t)$ is in M' , it must be supported by the new rule, *i.e.*, we must have $g(t) \leftarrow l, g(f(t))$, where l and $g(f(t))$ are true in M' . In particular, $M' \models l$. By deleting all g -literals from M' , we obtain a fixpoint model M of \mathbf{P} such that $M \models l$.

Statement (12) above is equivalent to $\text{comp}(\mathbf{P}) \models \neg l$, since the set of fixpoint models of \mathbf{P} coincides with the set of all models of $\text{comp}(\mathbf{P})$ [22]. Therefore, if we could decide superfiniteness, we could then determine, for each atom in the Herbrand universe of \mathbf{P} , whether or not $\text{comp}(\mathbf{P}) \models \neg l$. But then, since the Herbrand universe is recursively enumerable, this would be an algorithm for enumerating all negative ground atoms such that $\text{comp}(\mathbf{P}) \models \neg l$, contrary to the aforesaid Theorem 13 in [35].

□

Although many results exist regarding the various forms of finiteness, [16, 17, 20, 32, 36], several problems still remain open. The foremost among them is the question of whether query finiteness for Extended Datalog with FC's is decidable. This problem has inspired a number of studies [26, 19, 32], including the present work, but no solution has been found as of yet. It is also unknown whether superfiniteness is decidable for Extended Datalog with FD's. (As mentioned earlier, this problem is undecidable for the regular finiteness.) Furthermore, decidability of weak finiteness for Datalog with function symbols is also an open problem.

Abiteboul and Hull [1] have shown that a related problem of whether a given FD holds in a relation computed by a Datalog IDB from an EDB that satisfies certain FD's is undecidable. The answer to a similar question for FC's is unknown. The latter problem for FC's is closely related to decidability of finiteness for Extended Datalog with FC's. Indeed, if determining whether an FC holds in a relation computed by a Datalog IDB \mathbf{P} were recursively solvable, then we could decide whether $\text{Finite}(X)$ holds in the least model of \mathbf{P} , which is equivalent to finiteness. On the other hand, if we could prove that determining whether $\text{Finite}(X)$ holds is undecidable in the least model of \mathbf{P} (when arbitrary FC's are allowed to hold over the EDB-relations), then we would have shown undecidability of finiteness with FC's.

12 On Testing Query Finiteness

Examples 10.6 and 10.7 have demonstrated two important differences between finiteness and superfiniteness. First, finiteness is preserved under query equivalence, while superfiniteness is preserved only under uniform equivalence of [29]. Second, superfiniteness may fail to materialize when the query predicate can accommodate an infinite number of self-supporting facts, which is often caused by, so called, "cyclic facts" in the database (such as $p(a, a)$).

As remarked earlier, it is unknown whether finiteness is decidable, let alone axiomatizable. Nevertheless, our semi-naive algorithm, which is complete only for super-entailment of FC's, can be combined with other algorithms for FC-inference to yield stronger results. For instance, Kifer [16, 17] has shown that finiteness is decidable for extended Horn databases where the only FC's are of the form $\text{Finite}(i_1, \dots, i_k)$. Independently, Convent [7] proposed a similar decision procedure for the case when all EDB-predicates are finite, but IDBs need not be range-restricted.²² Sagiv and Vardi [32] developed a decision procedure for finiteness of *monadic* IDBs, *i.e.*, IDBs where all recursive predicates are unary.

To see how a combined procedure might work, we shall describe a slightly improved version of the algorithm from [16, 7, 17].

Algorithm 12.1 An improved version of the finiteness test from [16, 7, 17]

Input: *Horn IDB \mathbf{P} and a set \mathbf{F} of FC's of the form $d : \text{Finite}(i_1, \dots, i_k)$, where d is an EDB-predicate of \mathbf{P} .*

Output: *FC's of the form $p : \text{Finite}(j_1, \dots, j_m)$, where p is an IDB-predicate.*

²²Essentially, this amounts to considering range-restricted IDBs, where the only infinite EDB-relation is $\text{dom}(X)$, one that contains the entire domain.

Initialization: Construct an EDB where every relation contains one or more possibly non-ground tuples. If $d : \text{Finite}(i_1, \dots, i_k) \in \mathbf{F}$ then the corresponding relation, \bar{d} , has a tuple where positions i_1, \dots, i_k hold a distinguished constant \mathbf{a} ; other positions hold distinct variables that do not appear elsewhere in \bar{d} or in other tuples. Furthermore, each such tuple is replicated in \bar{d} (each time with new variables) for each body occurrence of d in \mathbf{P} .

Method: Evaluate the IDB bottom-up, starting with the EDB, until no new tuples can be generated (for non-ground tuples, a new tuple means it cannot be obtained by variable renaming from previously derived tuples). Let p be an arbitrary predicate and \bar{p} be the (non-ground) relation computed for p . If for some position, i , the projection $\bar{p}[i]$ contains no variables, then $p : i$ is a finite argument.

The IDB in Example 10.6 is easily handled by the above algorithm, as the only FC's there are of the form $\text{Finite}(\dots)$. To make things more interesting, we shall demonstrate the workings of Algorithm 12.1 on a more subtle example.

Example 12.2 (Finite, yet non-superfinite query) Let the query $?- q(X)$ be defined by the following IDB, where d is a finite predicate, *i.e.*, $\text{Finite}(X_1)$ and $\text{Finite}(Y_2)$ are input constraints:

$$\begin{aligned} p(X_1, Y_1) &\leftarrow d(X_1), f(Y_1) \\ p(X_2, Y_2) &\leftarrow f(X_2), d(Y_2) \\ p(X_3, Y_3) &\leftarrow p(X_3, Y_3) \\ q(X_4) &\leftarrow p(X_4, X_4) \end{aligned}$$

As in Example 10.6, q is non-superfinite because of the third, useless rule. If this rule were removed, it is easy to see that q would become superfinite and Algorithm 10.3 would be able to handle this case. However, Algorithm 12.1 can establish finiteness even in the presence of the third rule.

Algorithm 12.1 begins by initializing the relation for d to $\{\langle \mathbf{a} \rangle\}$ and the relation for f to $\{\langle V \rangle, \langle V' \rangle\}$. The bottom-up computation then derives $p(\mathbf{a}, V)$, $p(V, \mathbf{a})$, $p(\mathbf{a}, V')$, $p(V', \mathbf{a})$. Consequently, the extension of q is finite, as it contains exactly one tuple, $\langle \mathbf{a} \rangle$. \square

It is easy to modify the above example to show that Algorithms 10.3 and 12.1, when used in tandem, can detect query finiteness in cases where none of the methods can do this by itself. For instance, suppose that, in addition to the rules in Example 12.2, we had the following:

$$\begin{aligned} r(X_5) &\leftarrow g(X_5, Y_5), q(Y_5) \\ g : 2 \dashrightarrow 1 \end{aligned}$$

Then, since $q : \text{Finite}(1)$ has been inferred by Algorithm 12.1, it follows from the FC $g : 2 \dashrightarrow 1$ that r is finite. This example can be made arbitrarily complicated. For instance, we could plug q (along with its definition) into Example 10.5, where it would replace the finite EDB-predicate d . With this modification, the query predicate, p , becomes non-superfinite, but would still remain finite. Its finiteness is detectable by our combined algorithm.

13 Conclusions

We presented an axiomatization for superfiniteness and super-entailment of recursive Horn queries with infinite relations and finiteness constraints. This axiomatization yields an effective algorithm to decide the problem. The same machinery was then applied to the problem of inference of finiteness constraints over IDB-predicates in Horn databases—an important issue in processing queries with function symbols [30, 20, 17]. Although it is unknown whether entailment of finiteness constraints is decidable for Extended Datalog queries, we have shown that a stronger notion, super-entailment, is decidable. We have also shown how a decision procedure for super-entailment can enhance tests for query finiteness.

In the process, we developed a theory of finiteness and partial constraints, and investigated their interaction with inclusion, projection, and decomposition dependencies. Apart from the practical benefits mentioned earlier, this axiomatization has theoretical interest, since it is both very close to and fundamentally different from the inference problem for functional and inclusion dependencies, which is neither axiomatizable (in the classical sense) nor decidable.

Acknowledgments: I would like to thank Laks V.S. Lakshmanan and Shuky Sagiv for many stimulating discussions. Raghu Ramakrishnan and Avi Silberschatz helped at the early stages of this work. A very detailed report of one of the referees is responsible for the much improved presentation.

References

- [1] S. Abiteboul and R. Hull. Data functions, Datalog and negation. In *ACM SIGMOD Conference on Management of Data*, pages 143–154, New York, 1988. ACM.
- [2] K.R. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, Los Altos, CA, 1988.
- [3] C. Beeri and P.A. Bernstein. Computational problems related to the design of normal form relational schemes. *ACM Transactions on Database Systems*, 4(1):30–59, March 1979.
- [4] A. Brodsky and Y. Sagiv. On termination of Datalog programs. In *Intl. Conference on Deductive and Object-Oriented Databases*, pages 47–64. Elsevier Science Publ., 1989.
- [5] M.A. Casanova, R. Fagin, and C.H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *Journal of Computer and System Sciences*, 28:29–59, 1984.
- [6] A.K. Chandra and M.Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM Journal on Computing*, 14:671–677, 1985.
- [7] B. Convent. Deciding finiteness, groundedness and domain independence of pure Datalog queries. *J. of Information Processing and Cybernetics*, 25:401–416, 1989.

- [8] S.S. Cosmadakis and P.C. Kanellakis. Functional and inclusion dependencies: A graph theoretic approach. In P.C. Kanellakis, editor, *Advances in Computing Research*, volume 3, pages 163–184. Plenum Press, 1986.
- [9] P. De Bra. Horizontal decompositions based on functional-dependency-set implications. In *Intl. Conference on Database Theory*, volume 243 of *Lecture Notes in Computer Science*, pages 157–170, Rome, Italy, 1986. Springer-Verlag.
- [10] P. De Bra and J. Paredaens. Horizontal decompositions for handling exceptions to functional dependencies. In *Lecture Notes in Computer Science*, volume 154, pages 67–82. Springer-Verlag, 1983.
- [11] R.A. Di Paola. The recursive unsolvability of the decision problem for the class of definite formulas. *Journal of ACM*, pages 324–327, April 1969.
- [12] P.C. Fischer, J.H. Jou, and D.M. Tsou. Succinctness in dependency systems. *Theoretical Computer Science*, 24:323–329, 1983.
- [13] A N. Goodman and A O. Shmueli. Tree queries: A simple class of queries. *ACM Transactions on Database Systems*, pages 653–677, December 1982.
- [14] G. Gottlob. Computing covers for embedded functional dependencies. In *ACM Symposium on Principles of Database Systems*, pages 58–69, New York, March 1987. ACM.
- [15] D.S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28:167–189, 1984.
- [16] M. Kifer. On safety, domain independence, and capturability of database queries. In *3-d Intl. Conference on Data and Knowledge Bases*, pages 405–415, Jerusalem, Israel, June 1988. Morgan-Kaufmann.
- [17] M. Kifer. The relationship among finiteness, domain independence and capturability. Unpublished manuscript, 1990.
- [18] M. Kifer and E.L. Lozinskii. SYGRAF: Implementing logic programs in a database style. *IEEE Trans. on Software Engineering*, 14(7):922–935, 1988.
- [19] M. Kifer, R. Ramakrishnan, and A. Silberschatz. An axiomatic approach to deciding query safety in deductive databases. In *ACM Symposium on Principles of Database Systems*, pages 52–60, New York, March 1988. ACM.
- [20] R. Krishnamurthy, R. Ramakrishnan, and O. Shmueli. A framework for testing safety and effective computability. *Journal of Computer and System Sciences*, 52(1):100–124, February 1996.
- [21] V.S. Lakshmanan and D.A. Nonen. Superfiniteness of query answers in deductive databases: An automata-theoretic approach. In *12th Intl. Conference on Foundations of Software Technology and Theoretical Computer Science*, Dec 1992.

- [22] J.W. Lloyd. *Foundations of Logic Programming (Second Edition)*. Springer-Verlag, 1987.
- [23] D. Maier and D.S. Warren. *Computing with Logic: Logic Programming with Prolog*. Benjamin-Cummings, Menlo Park, CA, 1988.
- [24] J.C. Mitchell. The implication problem for functional and inclusion dependencies. *Information and Control*, 56:154–173, 1983.
- [25] S. Naqvi and S. Tsur. *A Logical Language for Data and Knowledge Bases*. Computer Science Press, Rockville, MD, 1989.
- [26] R. Ramakrishnan, F. Bancilhon, and A. Silberschatz. Safety of recursive horn clauses with infinite relations. In *ACM Symposium on Principles of Database Systems*, pages 328–339, New York, March 1987. ACM.
- [27] R. Ramakrishnan, D. Srivastava, and S. Sudarshan. CORAL: Control, relations and logic. In *Intl. Conference on Very Large Data Bases*, pages 238–250. Morgan Kaufmann, San Francisco, CA, August 1992.
- [28] R. Ramakrishnan, D. Srivastava, S. Sudarshan, and P. Seshadri. Implementation of the CORAL deductive database system. In *ACM SIGMOD Conference on Management of Data*, pages 167–176, Washington, D.C., May 1993. ACM.
- [29] Y. Sagiv. Optimizing Datalog programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698. Morgan Kaufmann, Los Altos, CA, 1988.
- [30] Y. Sagiv. On testing effective computability of magic programs. In *Intl. Conference on Deductive and Object-Oriented Databases*, pages 244–262, December 1991.
- [31] Y. Sagiv. A termination test for logic programs. In *Intl. Logic Programming Symposium*, pages 518–532, Cambridge, MA, November 1991. MIT Press.
- [32] Y. Sagiv and M.Y. Vardi. Safety of queries over infinite databases. In *ACM Symposium on Principles of Database Systems*, pages 160–171, New York, April 1989. ACM.
- [33] K. Sagonas, T. Swift, and D.S. Warren. XSB as an efficient deductive database engine. In *ACM SIGMOD Conference on Management of Data*, pages 442–453, New York, May 1994. ACM.
- [34] E. Sciore. Improving database schemes by adding attributes. In *ACM Symposium on Principles of Database Systems*, pages 379–383, New York, March 1983. ACM.
- [35] J.C. Shepherdson. Negation in logic programming. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 19–88. Morgan-Kaufmann, Los Altos, CA, 1988.
- [36] O. Shmueli. Decidability and expressiveness aspects of logic queries. In *ACM Symposium on Principles of Database Systems*, pages 237–249, New York, March 1987. ACM.
- [37] J.D. Ullman. *Principles of Database Systems*. Computer Science Press, Rockville, MD, 1982.

- [38] J.F. Ullman. *Principles of Database and Knowledge-Base Systems, Volume 1*. Computer Science Press, Rockville, MD, 1988.
- [39] J.F. Ullman. *Principles of Database and Knowledge-Base Systems, Volume 2*. Computer Science Press, Rockville, MD, 1989.
- [40] M.Y. Vardi. The decision problem for database dependencies. *Information Processing Letters*, pages 251–254, October 1981.
- [41] C. Zaniolo. Safety and compilation of non-recursive horn clauses. In *First Intl. Workshop on Expert Database Systems*, pages 63–73, Kiawah Island, South Carolina, October 1984.