On the expressive power of Query languages

Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Institut für
Informationssysteme

Peter Schäuble
Beat Wüthrich

On the Expressive Power of Query Languages

February 1992

173

Authors' addresses:

Peter Schäuble, Institut für Informationssysteme
ETH Zentrum, CH-8092 Zurich, Switzerland
e-mail: schauble@inf.ethz.ch

Beat Wüthrich, ECRC GmbH
Arabellastrasse 17, D-W-8000 München 81, Germany
e-mail: Beat.Wuetrich@ecrc.de

Abstract

Two main topics are addressed. First, an algebraic approach is presented to define a general notion of expressive power. Heterogeneous algebras represent information systems and morphisms represent the correspondences between the instances of databases, the correspondences between answers, and the correspondences between queries. An important feature of this new notion of expressive power is that query languages of different types can be compared with respect to their expressive power. In the case of relational query languages, the new notion of expressive power is shown to be equivalent to the notion used by Chandra and Harel. In the case of non-relational query languages, the versatility of the new notion of expressive power is demonstrated by comparing a Boolean query language with a weighted query language consisting of vague queries. We also demonstrate that the new notion of expressive power is appropriate to compare the fixpoint query languages with an object-oriented query language called FQL*. The expressive power of the Functional Query Language FQL* is the second main topic of this paper. The specifications of FQL* functions can be recursive or even mutually recursive. FQL* has a fixpoint semantics based on a complete lattice consisting of bag functions. The query language FQL* is shown to be more expressive than the fixpoint query languages. This result implies that FQL* is also more expressive than Datalog with stratified negation. Examples of recursive FQL* functions are given that determine the ancestors of persons and the bill of materials.

1 Introduction

The expressive power is one of the important features of a query language. Whether a specific need for information can be expressed in a query language or not depends on its expressive power. A thorough analysis of the expressive power of *relational query languages* has been presented by Chandra and Harel in [9], [10] (see also [11] for an overview and new results). They consider queries as partial functions mapping relational databases to relations. A query language is characterized by its corresponding set of queries. Considering a query language as a set of partial functions, a query language QL' is as expressive as a query language QL iff QL' is a superset of QL. Using this notion by Chandra and Harel, the expressive power of relational query languages can be represented by set diagrams. Figure 1 shows such a set diagram where the following query languages are represented as sets of partial functions: The first-order query language [12], Datalog [3], Datalog with stratified negation, and QPTIME, i.e. the set of relational queries that are evaluable in polynomial time [10]. For every atomic subset, a typical query is indicated. These queries are described below.

ForAll: Let $\mathcal{G}$ be any finite graph. Query: Given a vertex v, is there an edge from v to v' for all vertices v'?

Exists: Let $\mathcal{G}$ be any finite graph. Query: Given a vertex v, does at least one edge exist from v to any vertex v'?

TC (Transitive Closure): Let $\mathcal{G}$ be any finite graph. Query: Given a vertex v, which are the vertices v' that are reachable from v (i.e. there is a directed path from v to v')?

NotTC: Let $\mathcal{G}$ be any finite graph. Query: Given a vertex v, which are the vertices v' that are not reachable from v (i.e. there is no directed path from v to v')?

Game: Let $\mathcal{G}$ be any rooted tree where every leaf node is either black or white. There are two players who know $\mathcal{G}$ and the color of the leaf nodes. The root node is the start position of the game. Player I starts moving down the tree by one edge. Then, player II moves down another edge. They continue until a leaf node is reached. Player I wins if the leaf node is black and player II wins if the leaf node is white. Query: Does player I have a winning

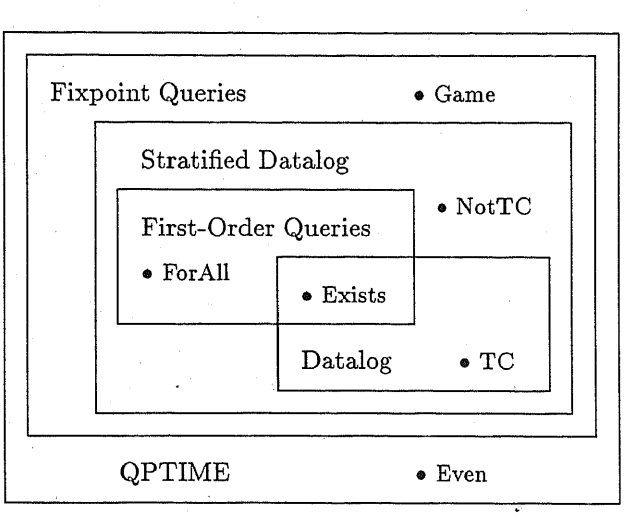

Figure 1: Expressive power of relational query languages according to Chandra and Harel.

strategy? Or equivalently, can player I win independently of the moves by player II?

Even: Let $\mathcal{G}$ be any finite graph. Query: Is the number of vertices an even number?

In Chandra and Harel's approach [10], a relational database $(D, \overline{R})$ of the type $\overline{a} = (a_0, \ldots, a_{k-1})$ consists of a domain D and a k-tuple $\overline{R} = (R_0, \ldots, R_{k-1})$ where $R_i \subseteq D^{a_i}$. The domain D is assumed to be a *finite subset* of a countable universe U. Chandra and Harel restrict themselves to queries of type $(\overline{a}, b)$, i.e. queries Q mapping a relational database $(D, \overline{R})$ of type $\overline{a}$ to a b-ary relation $Q(D, \overline{R}) \subseteq D^b$. From this restriction follows that there is no query to express the bill of materials as shown in the following. Let $D = \{0, ..., 2n-1\}$ be the domain and let R_0 be the relation defining the part/subpart structure shown in Figure 2. The parts are identified uniquely by numbers such that $D = \{p_1, \ldots, p_{n-1}, q_1, \ldots, q_{n-1}, r, s\}$. We define the following costs:

$$\begin{aligned} cost(r) &= 0 \\ cost(s) &= 1 \end{aligned}$$

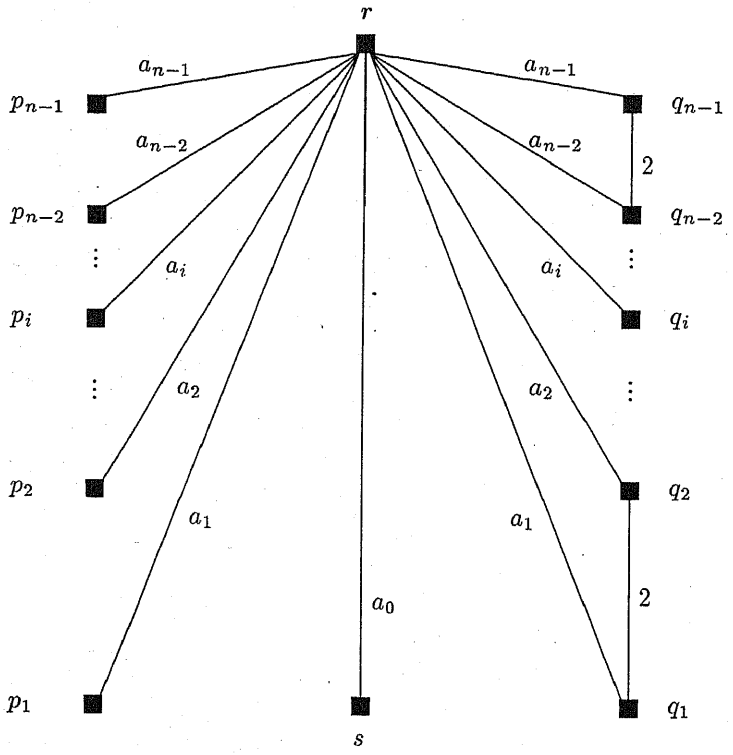

Figure 2: Given this part/subpart structure, the total cost of part r and its subparts cannot be expressed by any query considered by Chandra and Harel.

$$\begin{aligned} cost(p_i) &= 1 \qquad 0 < i < n \\ cost(q_i) &= 1 \qquad 0 < i < n \end{aligned}$$

The part q_i is contained twice in q_{i-1}. The parts p_i and q_i are contained a_i times in r where $a_i \in \{0, 1\}$. Similarly, the part s is contained a_0 times in r. The cost of r and all its subparts are

$$total\_cost(r) = \sum_{i=0}^{n-1} a_i 2^i$$

as it can easily be shown by induction on n. Thus, depending on the part/subpart structure, i.e. depending on $a_0, \ldots, a_{n-1}$, the cost are

$$total\_cost(r) \in \{0, ..., 2^n - 1\}$$

and hence, the domain $D = \{0, ..., 2n - 1\}$ does not contain enough elements to represent every possible answer if $n > 2$. More generally speaking, query languages having the ability to count cannot be analyzed within the framework of [10].

In this paper, we present a more general framework to study the expressive power of arbitrary query languages. We rely on an algebraic approach where heterogeneous structures represent information systems and morphisms represent the correspondences between the instances of the databases, the correspondences between the answers, and the correspondences between the queries. Let IS_1 and IS_2 be two structures, each representing an information system. Such an information system consists of a query language, a set of database instances, a set of answers, and a query evaluation function. We call the query language of IS_2 as expressive as the query language of IS_1 iff there exists a morphism from IS_1 to IS_2. In Chandra and Harel's approach "as expressive as" is defined by means of set inclusion whereas in our approach, "as expressive as" is defined by means of morphisms. This kind of generalization is often used in mathematics. In our case, this generalization facilitates the comparison of the expressive power of query languages of *different types*. This more general comparison capability is demonstrated by comparing a Boolean query language and a weighted query language consisting of vague queries. The answer to a Boolean query consists of a set of data items, whereas the answer to a weighted query consists of a function which assigns every data item a probability that the data item belongs to the answer. We shall present

a necessary and sufficient condition for the cases where the Boolean query language is less expressive than the weighted query language.

A further generalization consists of allowing infinite domains. It should be noted that infinite domains are not only important for counting but also for vague queries [15], [23] and for Information Retrieval query languages [22], [27]. Infinite domains do not necessarily lead to unbounded computation times. This fact is demonstrated by a query language FQL* which is more expressive than the fixpoint query languages but still evaluable in polynomial time.

FQL* is a Functional Query Language with a syntax similar to the syntax of OSQL [6]. OSQL is an interactive interface to the Iris DBMS [28]. Unlike the specifications of OSQL functions, the specifications of FQL* functions can be recursive or even mutually recursive. FQL* has a fixpoint semantics based on a complete lattice consisting of bag functions. Examples of recursive FQL* functions are given that determine the ancestors of persons and the bill of materials. It is shown that these FQL* functions are computed in polynomial time even if the parents relation and the parts_of relation contain cycles. In these cases, a person may be an ancestor of himself and the cost of a part may be infinite. We show that there are non-hierarchical relations where these results are meaningful.

The main contributions of this paper are the following. First of all, a new notion of expressive power is introduced which facilitates the comparison of formal languages with respect to their expressive power even when these query languages and their underlying data model are of *different* types. Second, a new criterion is presented which characterizes the cases when a Boolean query language is less expressive than a weighted query language. Finally, an object-oriented query language is introduced which has an expressive power that goes beyond the expressive power of the fixpoint query languages.

The paper is organized as follows. In Section 2, our new notion of expressive power is introduced. In Section 3, we show that for relational query languages, Chandra and Harel's notion of expressive power coincides with our notion of expressive power. In Section 4, the expressive power of a Boolean query language and the expressive power of a weighted query language are compared. In Section 5, the object-oriented query language FQL* is introduced. In Section 6, we discuss the expressive power of FQL*. In Section 7, some conclusions are drawn. Appendix A contains the proofs of two theorems.

2 A New Notion of Expressive Power

In this section, we introduce a new notion of expressive power. We start with a simple model of an information system. An information system is represented by a structure, i.e. a heterogeneous algebra [7]

$$IS \quad := \quad \langle DBS, QL, AS, \alpha \rangle \tag{1}$$

where the database system, DBS, is the set of possible instances of the database; the query language, QL, is the set of possible queries; the answer system, AS, is the set of possible answers; the query evaluation function, α, is a partial function mapping $DBS \times QL$ to AS. The following example shows an information system based on a relational database system that is accessible by means of relational algebra.

Example 1 *The components of the information system*

$$IS_{ra} \quad = \quad \langle DBS_{rel}, QL_{ra}, AS_{rel}, \alpha_{ra} \rangle$$

are the following. The database system DBS_{rel} consists of all instances of a relational database. Every instance $(D, \overline{R})$ is represented by a domain D and tuple of relations $\overline{R} = (R_0, \ldots, R_{k-1})$ where $R_i \subseteq D^{a_i}$. The query language QL_{ra} consists of relational algebra expressions [26]. The answer system AS_{rel} consists of relations $R \subseteq D^b$. The evaluation function α_{ra} assigns every instance $(D, \overline{R})$ of the database and every query q an answer $\alpha_{ra}((D, \overline{R}), q)$.

A complete specification of an information system should also contain the specification of a *modification language* ML and the specification of a *modification function* $\mu : DBS \times ML \rightarrow DBS$. Since, in this paper, we are interested only in the expressive power of the query language, it is sufficient to represent an information system by the simplified structure (1) which defines neither a modification language nor a modification function.

Below, we define when a query language QL_2 is as expressive as a query language QL_1. We will see that this relationship between QL_1 and QL_2 depends on (1) the information systems to which QL_1 and QL_2 belong, (2) the correspondences between the instances $db_1 \in DBS_1$ and the instances $db_2 \in DBS_2$, and (3) the correspondences between the answers $a_1 \in AS_1$ and the answers $a_2 \in AS_2$. The correspondences between the instances of the databases are given by the function f, the correspondences between the answers are given

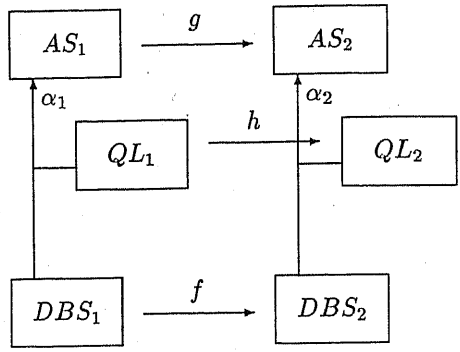

Figure 3: Expressive power of query languages.

by the function g, and the correspondences between the queries are represented by the function h (Figure 3).

Definition 1 *Let* $IS_1 = \langle DBS_1, QL_1, AS_1, \alpha_1 \rangle$ *and* $IS_2 = \langle DBS_2, QL_2, AS_2, \alpha_2 \rangle$ *be two information systems and let* $f : DBS_1 \rightarrow DBS_2$ *be a function which determines for every instance* $db_1 \in DBS_1$ *a corresponding instance* $f(db_1) \in DBS_2$ *and let* $g : range(\alpha_1) \rightarrow range(\alpha_2)$ *be an injective function which determines for every answer* $a_1 \in range(\alpha_1)$ *a corresponding answer* $g(a_1) \in range(\alpha_2)$. *The query language* QL_2 *of* IS_2 *is called as expressive as the query language* QL_1 *of* IS_1 *with respect to the correspondences given by* f *and* g *iff there exists a function* $h : QL_1 \rightarrow QL_2$ *such that*

$$\forall (db_1, q_1) \in dom(\alpha_1) : g(\alpha_1(db_1, q_1)) = \alpha_2(f(db_1), h(q_1)). \qquad (2)$$

The function g is required to be injective because two different answers of AS_1 must correspond to two different answers of AS_2. The function f is not required to be injective because several equivalent instances in DBS_1 may correspond to a single instance in DBS_2 (see below). Subsequently,

$$QL_1(IS_1) \quad \leq_{f,g} \quad QL_2(IS_2) \qquad (3)$$

denotes that QL_2 of IS_2 is as expressive as QL_1 of IS_1 with respect to the correspondences given by f and g. The relationship $\leq_{f,g}$ can also be characterized by means of morphisms. Morphisms are defined as follows.

Definition 2 *Let $\mathcal{S} = \langle U, V, W, \alpha \rangle$ and $\mathcal{T} = \langle X, Y, Z, \beta \rangle$ be two structures where $\alpha : U \times V \to W$ and $\beta : X \times Y \to Z$ are partial functions. The set of functions $\{f : U \to X, g : W \to Z, h : V \to Y\}$ is called a morphism if*

$$\forall (u, v) \in dom(\alpha) : g(\alpha(u, v)) = \beta(f(u), h(v)) \tag{4}$$

If the functions f, g, and h are bijective the morphism $\{f, g, h\}$ is called an isomorphism [7].

Proposition 1 *Let $IS_1 = \langle DBS_1, QL_1, AS_1, \alpha_1 \rangle$ and $IS_2 = \langle DBS_2, QL_2, AS_2, \alpha_2 \rangle$ be two information systems and let $f : DBS_1 \to DBS_2$ be a function which determines for every instance $db_1 \in DBS_1$ a corresponding instance $f(db_1) \in DBS_2$ and let $g : range(\alpha_1) \to range(\alpha_2)$ be an injective function which determines for every answer $a_1 \in range(\alpha_1)$ a corresponding answer $g(a_1) \in range(\alpha_2)$. The query language QL_2 of IS_2 is as expressive as the query language QL_1 of IS_1 with respect to the correspondences given by f and g iff there exists a function $h : QL_1 \to QL_2$ such that $\{f : DBS_1 \to DBS_2, g : range(\alpha_1) \to range(\alpha_2), h : QL_1 \to QL_2\}$ is a morphism from IS_1 to IS_2.*

Proof. $\{f, g, h\}$ is a morphism because of (2). Conversely, (2) is satisfied because $\{f, g, h\}$ is a morphism. $\diamondsuit$

In order to define "equally expressive", we introduce an equivalence relation between the instances of the database and an equivalence relation between the queries. Given an information system $IS = \langle DBS, QL, AS, \alpha \rangle$, two instances $db, db' \in DBS$ are called *equivalent* iff for every query $q \in QL$, the answer $\alpha(db, q)$ is equal to the answer $\alpha(db', q)$. Similarly, two queries $q, q' \in QL$ are called *equivalent* iff for every instance $db \in DBS$, the answer $\alpha(db, q)$ is equal to the answer $\alpha(db, q')$.

$$db \approx db' \quad :\Longleftrightarrow \quad \forall q \in QL : \alpha(db, q) = \alpha(db', q) \tag{5}$$

$$q \approx q' \quad :\Longleftrightarrow \quad \forall db \in DBS : \alpha(db, q) = \alpha(db, q') \tag{6}$$

Example 2 *Let $\langle DBS_{rel}, QL_{ra}, AS_{rel}, \alpha_{ra} \rangle$ be an information system consisting of a relational database system accessible through relational algebra (Example 1). The following queries are equivalent.*

$$select[age = 20](project[name, age](employee))$$
$$project[name, age](select[age = 20](employee))$$

It is easy to show that both the relation $\approx$ defined on DBS and the relation $\approx$ defined on QL are equivalence relations, i.e. reflexive, symmetric, and transitive relations. Hence, the set DBS and the set QL are partitioned into disjoint equivalence classes. The class consisting of all instances that are equivalent to the instance db is denoted by $[db]$. Similarly, the class consisting of all queries that are equivalent to the query q is denoted by $[q]$. The set of all equivalence classes of instances in DBS is denoted by $DBS^{\approx}$ and the set of all equivalence classes of queries of QL is denoted by $QL^{\approx}$. Using this notation, every evaluation function α determines a function $\alpha^{\approx}$.

$$\alpha^{\approx} : DBS^{\approx} \times QL^{\approx} \to AS,\ ([db],[q]) \mapsto \alpha(db,q) \tag{7}$$

Note that $\alpha^{\approx}$ is well defined. From $db \approx db'$ and $q \approx q'$ follows that $\alpha(db,q) = \alpha(db',q')$ and hence, the function value $\alpha^{\approx}([db],[q])$ is independent from the particular instance db representing $[db]$ and from the particular query q representing $[q]$. Every function $f : DBS_1 \to DBS_2$ satisfying

$$db_1 \approx db_1' \implies f(db_1) \approx f(db_1') \tag{8}$$

induces a mapping $f^{\approx}$ from $DBS_1^{\approx}$ to $DBS_2^{\approx}$.

$$f^{\approx} : DBS_1^{\approx} \to DBS_2^{\approx}, [db_1] \mapsto [f(db_1)] \tag{9}$$

Note that the function value $f^{\approx}([db_1])$ is well defined. Because of (8) it is independent of the particular instance db_1 representing the equivalence class $[db_1]$.

The relationship "equally expressive" can be defined in two different ways. First, it can be defined by means of an isomorphisms from $\langle DBS_1^{\approx}, QL_1^{\approx}, AS_1, \alpha_1^{\approx}\rangle$ to $\langle DBS_2^{\approx}, QL_2^{\approx}, AS_2, \alpha_2^{\approx}\rangle$. Second, it can be defined by means of $\leq_{f,g}$ and $\geq_{f^*,g^{-1}}$ provided the correspondences given by $f : DBS_1 \to DBS_2$ and the correspondences given by $f^* : DBS_2 \to DBS_1$ are compatible (i.e. $f^*(f(db_1)) \approx db_1$ and $f(f^*(db_2)) \approx db_2$). Figure 4 shows that this kind of compatibility requires a bijective correspondence between the equivalence classes rather than between the particular instances of the two databases. For instance, the equivalence class $\{db_1, db_1', db_1''\}$ corresponds to the class $\{db_2, db_2'\}$ and there exists no bijective correspondence between the elements of the classes.

We will use isomorphisms, i.e. the first alternative, to define the relationship "equally expressive." Later, we will show that this def-

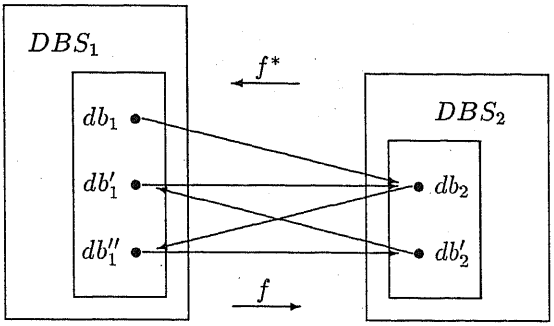

Figure 4: Correspondences between equivalence classes.

inition is equivalent to the second alternative (Proposition 4). This shows that the concept of "as expressive as" is quite a robust concept.

Definition 3 *Let* $IS_1 = \langle DBS_1, QL_1, AS_1, \alpha_1 \rangle$ *and* $IS_2 = \langle DBS_2, QL_2, AS_2, \alpha_2 \rangle$ *be two information systems. Assume that the correspondences between the instances of the databases and between the answers are given by the function* $f : DBS_1 \rightarrow DBS_2$ *and by the function* $g : AS_1 \rightarrow AS_2$ *respectively.* QL_1 *of* IS_1 *and* QL_2 *of* IS_2 *are called equally expressive with respect to* f *and* g*, denoted by*

$$QL_1(IS_1) \quad =_{f,g} \quad QL_2(IS_2), \tag{10}$$

iff f *satisfies (8) and there exists a function* $h^{\approx} : QL_1^{\approx} \rightarrow QL_2^{\approx}$ *such that the morphism* $\{f^{\approx}, g, h^{\approx}\}$ *is an isomorphism from* $\langle DBS_1^{\approx}, QL_1^{\approx}, AS_1, \alpha_1^{\approx} \rangle$ *to* $\langle DBS_2^{\approx}, QL_2^{\approx}, AS_2, \alpha_2^{\approx} \rangle$.

The existence of an isomorphism $\{f, g, h\}$ from the information system $\langle DBS_1, QL_1, AS_1, \alpha_1 \rangle$ to the information system $\langle DBS_2, QL_2, AS_2, \alpha_2 \rangle$ implies the existence of an isomorphism $\{f^{\approx}, g, h^{\approx}\}$ from $\langle DBS_1^{\approx}, QL_1^{\approx}, AS_1, \alpha_1^{\approx} \rangle$ to $\langle DBS_2^{\approx}, QL_2^{\approx}, AS_2, \alpha_2^{\approx} \rangle$ but not vice versa. This asymmetry is because two equivalent instances db_1 and db'_1 in DBS_1 may correspond to a single instance db_2 in DBS_2 (see Figure 4) or two equivalent queries q_1 and q'_1 in QL_1 may correspond to a single query q_2 in QL_2. Thus, two query languages can be equally expressive even if there are no bijective correspondences between the queries or between the instances of the databases.

Definition 4 *QL_2 of IS_2 is more expressive than QL_1 of IS_1 with respect to f and g, denoted by*

$$QL_1(IS_1) \quad <_{f,g} \quad QL_2(IS_2), \tag{11}$$

iff QL_2 is as expressive as QL_1 but QL_1 and QL_2 are not equally expressive.

The following proposition will be useful in the subsequent sections where we will prove that some query languages are more expressive than others.

Proposition 2 *If $QL_1(IS_1) =_{f,g} QL_2(IS_2)$, there exists for every query $q_2 \in QL_2$ a query $q_1 \in QL_1$ such that for all $db_1 \in DBS_1$,*

$$(db_1, q_1) \in domain(\alpha_1) \quad \Rightarrow \quad g(\alpha_1(db_1, q_1)) = \alpha_2(f(db_1), q_2) \tag{12}$$

Proof. From $QL_1(IS_1) =_{f,g} QL_2(IS_2)$ it follows that every query q_2 belongs to an equivalence class $[h(q_1)]$. From (4) follows that q_1 satisfies (12). $\Diamond$

If $QL_1(IS_1) \leq_{f,g} QL_2(IS_2)$ and we want to prove that $QL_1(IS_1) <_{f,g} QL_2(IS_2)$, it is sufficient to show that (12) yields a contradiction. The following proposition shows the transitivity of the relationships $\leq_{f,g}$, $=_{f,g}$, and $<_{f,g}$.

Proposition 3 *Given $\theta \in \{\leq, =, <\}$, from $QL_1(IS_1)\ \theta_{f,g}\ QL_2(IS_2)$ and $QL_2(IS_2)\ \theta_{f',g'}\ QL_3(IS_3)$ follows $QL_1(IS_1)\ \theta_{f' \circ f, g' \circ g}\ QL_3(IS_3)$.*

Proof. The transitivity of $\leq_{f,g}$, $=_{f,g}$, and $<_{f,g}$ follows immediately from their definitions. $\Diamond$

Finally, we will show that $=_{f,g}$ is equivalent to $\leq_{f,g}$ and $\geq_{f^*,g-1}$ provided the correspondences given by f and the correspondences given by f^* are compatible.

Proposition 4 *Let $IS_1 = \langle DBS_1, QL_1, AS_1, \alpha_1 \rangle$ and $IS_2 = \langle DBS_2, QL_2, AS_2, \alpha_2 \rangle$ be two information systems. Assume that the correspondences between the instances of the databases and between the answers are given by the function $f : DBS_1 \rightarrow DBS_2$ and by the function $g : AS_1 \rightarrow AS_2$ respectively. Then, $QL_1(IS_1) =_{f,g} QL_2(IS_2)$ iff there is a function $f^* : DBS_2 \rightarrow DBS_1$ compatible with f (i.e. $\forall db_2 \in DBS_2 : f(f^*(db_2)) \approx db_2$ and $\forall db_1 \in DBS_1 : f^*(f(db_1)) \approx db_1$) such that $QL_1(IS_1) \leq_{f,g} QL_2(IS_2)$ and $QL_1(IS_1) \geq_{f^*,g^{-1}} QL_2(IS_2)$.*

Proof. From $QL_1(IS_1) =_{f,g} QL_2(IS_2)$ it follows that $QL_1(IS_1) \leq_{f,g} QL_2(IS_2)$ because for every $(db_1, q_1) \in range(\alpha_1)$,

$$\begin{aligned} g(\alpha_1(db_1, q_1)) &= g(\alpha_1^{\approx}([db_1], [q_1])) \\ &= \alpha_2^{\approx}(f^{\approx}([db_1]), h^{\approx}([q_1])) \\ &= \alpha_2(f(db_1), h(q_1)). \end{aligned}$$

Assume that $QL_1(IS_1) =_{f,g} QL_2(IS_2)$. Since $\{f^{\approx}, g, h^{\approx}\}$ is an isomorphism, there exists a function $f^* : DBS_2 \rightarrow DBS_1$ such that $\forall db_2 \in DBS_2 : f(f^*(db_2)) \approx db_2$ and $\forall db_1 \in DBS_1 : f^*(f(db_1)) \approx db_1$ (see also Figure 4). For the same reasons, there exists a function $h^* : QL_2 \rightarrow QL_1$ such that $\forall q_2 \in QL_2 : h(h^*(q_2)) \approx q_2$ and $\forall q_1 \in QL_1 : h^*(h(q_1)) \approx q_1$. For every $(db_2, q_2) \in range(\alpha_2)$, we obtain the equalities

$$\begin{aligned} g(\alpha_1(f^*(db_2), h^*(q_2))) &= g(\alpha_1^{\approx}([f^*(db_2)], [h^*(q_2)])) \\ &= \alpha_2^{\approx}(f^{\approx}([f^*(db_2)]), h^{\approx}([h^*(q_2)])) \\ &= \alpha_2(f(f^*(db_2)), h(h^*(q_2))) \\ &= \alpha_2(db_2, q_2). \end{aligned}$$

Since g is bijective, $\alpha_1(f^*(db_2), h^*(q_2)) = g^{-1}(\alpha_2(db_2, q_2))$ and hence, $QL_1(IS_1) \geq_{f^*, g^{-1}} QL_2(IS_2)$.

The other direction is proven as follows. Assume that there exists a function $f^* : DBS_2 \rightarrow DBS_1$ such that $\forall db_2 \in DBS_2 : f(f^*(db_2)) \approx db_2$, $\forall db_1 \in DBS_1 : f^*(f(db_1)) \approx db_1$, $QL_1(IS_1) \leq_{f,g} QL_2(IS_2)$, and $QL_1(IS_1) \geq_{f^*, g^{-1}} QL_2(IS_2)$. The function $f^{\approx}$ is well defined because

$$\begin{aligned} & f(db_1) \not\approx f(db_1') \\ & \quad \Longrightarrow \exists q_2 : \alpha_2(f(db_1), q_2) \neq \alpha_2(f(db_1'), q_2) \\ & \quad \Longrightarrow \exists q_2 : g^{-1}(\alpha_2(f(db_1), q_2)) \neq g^{-1}(\alpha_2(f(db_1'), q_2)) \\ & \quad \Longrightarrow \exists q_2 : \alpha_1(f^*(f(db_1)), h^*(q_2)) \neq \alpha_1(f^*(f(db_1')), h^*(q_2)) \\ & \quad \Longrightarrow \exists q_1 : \alpha_1(db_1, q_1) \neq \alpha_1(db_1', q_1) \\ & \quad \Longrightarrow db_1 \not\approx db_1'. \end{aligned}$$

This implication is equivalent to (8). The function $f^{\approx}$ is injective because

$$db_1 \not\approx db_1' \implies \exists q_1 : \alpha_1(db_1, q_1) \neq \alpha_1(db_1', q_1)$$

$$
\begin{aligned}
&\Longrightarrow \quad \exists q_1 : g(\alpha_1(db_1, q_1)) \neq g(\alpha_1(db_1', q_1)) \\
&\Longrightarrow \quad \exists q_1 : \alpha_2(f(db_1), h(q_1))) \neq \alpha_2(f(db_1'), h(q_1))) \\
&\Longrightarrow \quad f(db_1) \not\approx f(db_1').
\end{aligned}
$$

The function $f^{\approx}$ is surjective because for $\forall db_2 \in DBS_2 : f(f^*(db_2)) \approx db_2$. This shows that $f^{\approx}$ is bijective.

Since g^{-1} exists, the function g is bijective. The function $h^{\approx}$ is well defined because

$$
\begin{aligned}
q_1 \approx q_1' \quad &\Longleftrightarrow \quad \forall db_1 : \alpha_1(db_1, q_1) = \alpha_1(db_1, q_1') \\
&\Longleftrightarrow \quad \forall db_1 : g(\alpha_1(db_1, q_1)) = g(\alpha_1(db_1, q_1')) \\
&\Longleftrightarrow \quad \forall db_1 : \alpha_2(f(db_1), h(q_1)) = \alpha_2(f(db_1), h(q_1')) \\
&\Longleftrightarrow \quad \forall db_2 : \alpha_2(db_2, h(q_1)) = \alpha_2(db_2, h(q_1')) \\
&\Longleftrightarrow \quad h(q_1) \approx h(q_1').
\end{aligned}
$$

We used the fact that g is injective and $f^{\approx}$ is surjective. At the same time, we can conclude that $h^{\approx}$ is injective.

The function $h^{\approx}$ is surjective because $q_2 \in QL_2$, we have for every $db_2 \in DBS_2$:

$$
\begin{aligned}
\alpha_2(db_2, q_2) &= g(\alpha_1(f^*(db_2), h^*(q_2))) \\
&= \alpha_2(f(f^*(db_2)), h(h^*(q_2))) \\
&= \alpha_2(db_2, h(h^*(q_2))).
\end{aligned}
$$

and hence, $h(h^*(q_2)) \approx q_2$. This shows that $h^{\approx}$ is bijective.

Because $\{f, g, h\}$ is a morphism, $\{f^{\approx}, g, h^{\approx}\}$ is also a morphism. It is actually an isomorphism because $f^{\approx}$, g, and $h^{\approx}$ are bijective and hence, $QL_1(IS_1) =_{f,g} QL_2(IS_2)$. $\diamondsuit$

In this section, we have defined "as expressive as" ($\leq_{f,g}$), "equally expressive" ($=_{f,g}$), and "more expressive than" ($<_{f,g}$). The first relationship is associated with a morphism from one information system to another information system. The second relationship is associated with an isomorphism from the first information system modulo $\approx$ to the second information system modulo $\approx$. Alternatively, $=_{f,g}$ can be defined as $\leq_{f,g}$ and $\geq_{f^*,g^{-1}}$. The third relationship is defined as usual, i.e. as $\leq_{f,g}$ and $\neq_{f,g}$. In the subsequent sections, such relationships will be identified between query languages of various types.

3 Relational Query Languages

In this section, we first define what relational query languages are, in the sense of Chandra and Harel [9], [10]. We will show that when comparing the expressive power of relational query languages, Chandra and Harel's notion of expressive power is equivalent to our notion of expressive power.

An instance $(D, \overline{R})$ of a relational database system DBS_{rel} consists of a finite domain D and relations $\overline{R} = (R_0, \ldots, R_{k-1})$ where $R_i \subseteq D^{a_i}$. The countable universe U is assumed to entail all domains.

$$\forall (D, \overline{R}) \in DBS_{rel} : D \subseteq U \wedge |D| \in \mathsf{N} \tag{13}$$

The answer system contains all relations that have a finite arity.

$$AS_{rel} \quad := \quad \{R \subseteq U^b \mid b \in \mathsf{N}\} \tag{14}$$

Definition 5 (Chandra & Harel) *Given an information system* $IS = \langle DBS_{rel}, QL, AS_{rel}, \alpha \rangle$, *the query language* QL *is called relational iff for every query* $q \in QL$, *there exist a tuple* $\overline{a} = (a_0, \ldots, a_{k-1}) \in \mathsf{N}^k$ *and a natural number* $b \in \mathsf{N}$ *such that from* $((D, \overline{R}), q) \in dom(\alpha)$ *it follows that* $\overline{R} = (R_0, \ldots, R_{k-1})$ *where* $R_i \subseteq D^{a_i}$ *and* $\alpha((D, \overline{R}), q) \subseteq D^b$. *The tuple* $(\overline{a}, b)$ *is called the type of the query* q. *The query* q *is called computable iff the function*

$$F_q : DBS_{rel} \rightarrow AS_{rel}, (D, \overline{R}) \mapsto \alpha((D, \overline{R}), q) \tag{15}$$

is a partial recursive function and for every isomorphism h *from* $(D, \overline{R})$ *to* $(D', \overline{R}')$, $\overline{h}(F_q(D, \overline{R})) = F_q(D', \overline{R}')$ *where* $\overline{h}(x_0, \ldots, x_{b-1}) := (h(x_0), \ldots, h(x_{b-1}))$.

As we pointed out in the introduction, an important consequence of this definition is that relational query languages cannot count. For instance, the query that computes the bill of materials (Figure 2) does not belong to a relational query language because its range is infinite. Below, a fixpoint query language is given as an example of such a relational query language. There are other fixpoint query languages which all have the same expressive power [18], [20].

The fixpoint query language $FO + LFP$ encompasses a fixpoint operator. Let $(D, \overline{R})$ be an instance of a relational database as described above. Furthermore, let $\varphi(P, \overline{x})$ be a first-order formula with a distinguished r-ary predicate symbol P and r free variables $\overline{x} =$

$(x_0, \ldots, x_{r-1})$. In addition to P and $\overline{x}$, the formula $\varphi(P, \overline{x})$ may also contain $\wedge$, $\vee$, $\neg$, $\exists$, $\forall$, $R_0 \ldots, R_{k-1}$, and bound variables. The formula $\varphi(P, \overline{x})$ is assumed to contain only positive occurrences of P. Thus, the operator τ given by $\tau(P) := \{\overline{d} \in D^r \mid \varphi(P, \overline{d})\}$ has a least fixpoint $\varphi_\infty := lfp(\tau)$ because τ is monotonic and D is finite. When the first-order query language FO is augmented by a fixpoint operator, the fixpoint query language $FO + LFP$ is obtained which is much more expressive than the first-order query language FO (Figure 1). The fixpoint operator determines the semantics of P occurring in a first-order formula $\varphi(P, \overline{x})$. The fixpoint query language $FO + LFP$ can be restricted to formulas containing the fixpoint operator at most once [20].

In what follows, two examples of fixpoint queries are given. First, when the fixpoint operator is applied to

$$\varphi(P, (x_0, x_1)) \quad := \quad R(x_0, x_1) \vee (\exists x)(R(x_0, x) \wedge P(x, x_1)), \quad (16)$$

the transitive closure of R is obtained (query TC in Figure 1). Second, applying the fixpoint operator to

$$\psi(P, (x_0)) \quad (17)$$
$$:= \quad B(x_0) \vee (\exists x)(E(x_0, x) \wedge (\forall x')(E(x, x') \rightarrow P(x'))) \quad (18)$$

yields a least fixpoint which determines whether player I has a winning strategy or not. The relation E represents the edges of the tree and B represents the black leave nodes. Let ψ_∞ be the least fixpoint and r be the root of the tree. Then, $\psi_\infty(r)$ is the query *Game* in Figure 1.

The query *Game* cannot be formulated in Datalog with stratified negation [10]. However, if the body of the rules may contain quantifiers [29], the query *Game* can be expressed by means of the following rules.

$$win(S) \quad \leftarrow \quad black(S). \quad (19)$$
$$win(S) \quad \leftarrow \quad move(S, T) \wedge \forall U : (move(T, U) \rightarrow win(U)). \quad (20)$$

Let QL_1 and QL_2 be two relational query languages which belong to the information systems $IS_1 = \langle DBS_{rel}, QL_1, AS_{rel}, \alpha_1 \rangle$ and $IS_2 = \langle DBS_{rel}, QL_2, AS_{rel}, \alpha_2 \rangle$ respectively. According to Chandra and Harel, QL_2 is as expressive as QL_1 iff $\{F_{q_1} \mid q_1 \in QL_1\} \subseteq \{F_{q_2} \mid q_2 \in QL_2\}$. The following proposition shows that this notion of expressive power is equivalent to our notion of expressive power.

Proposition 5 *Let* $IS_1 = \langle DBS, QL_1, AS, \alpha_1 \rangle$ *and* $IS_2 = \langle DBS, QL_2, AS, \alpha_2 \rangle$ *be two information systems that are based on the same database system and on the same answer system. Furthermore, let* $I_1 : DBS \to DBS$ *and* $I_2 : AS \to AS$ *be the identity mappings, i.e.* $I_1(db) = db$ *and* $I_2(a) = a$. *Then,* $QL_1(IS_1) \leq_{I_1,I_2} QL_2(IS_2)$ *iff* $\mathcal{F}_1 := \{F_{q_1} : DBS \to AS, db \mapsto \alpha_1(db, q_1) \mid q_1 \in QL_1\} \subseteq \mathcal{F}_2 := \{F_{q_2} : DBS \to AS, db \mapsto \alpha_2(db, q_2) \mid q_2 \in QL_2\}$.

Proof. From $QL_1(IS_1) \leq_{I_1,I_2} QL_2(IS_2)$ follows $F_{q_1} = F_{h(q_1)}$ and hence, $\mathcal{F}_1 \subseteq \mathcal{F}_2$. Conversely, from $\mathcal{F}_1 \subseteq \mathcal{F}_2$ follows that for every $q_1 \in QL_1$ there exists a $q_2 \in QL_2$ such that $F_{q_1} = F_{q_2}$. Hence, there exists a mapping $h : QL_1 \to QL_2$ such that $F_{q_1} = F_{h(q_1)}$ for all $q_1 \in QL_1$. The existence of such a mapping implies that $QL_1(IS_1) \leq_{I_1,I_2} QL_2(IS_2)$. $\diamondsuit$

Using the proposition above, the results of various authors on the expressive power of relational query languages can be summarized in the following way (see also Figure 1).

$$FO =_{I_1,I_2} RelAlg \tag{21}$$

$$FO <_{I_1,I_2} Datalog^{\neg} \tag{22}$$

$$Datalog <_{I_1,I_2} Datalog^{\neg} \tag{23}$$

$$Datalog^{\neg} <_{I_1,I_2} FO + LFP \tag{24}$$

$$FO + LFP <_{I_1,I_2} QPTIME \tag{25}$$

First-order queries $\varphi \in FO$ are first-order formulas. The answer to a first-order query φ is $\{\overline{d} \mid \varphi(\overline{d})\}$. As shown in [12], FO and relational algebra, $RelAlg$, are equally expressive. If $<$, $>$, $\leq$, and $\geq$ are excluded, FO is less expressive than $Datalog^{\neg}$, i.e. Datalog with stratified negation [3]. This fact is shown in [1]. Whether or not the transitive closure can be expressed in $RelAlg$ or in FO when $<$, $>$, $\leq$, and $\geq$ are available seems to be an open question (see also comment in [17]). Negation is not allowed in pure $Datalog$ [3]. Due to the missing negation, the query $NotTC$ is not expressible in $Datalog$ (Figure 1). Thus, $Datalog^{\neg}$ is more expressive than pure $Datalog$. Furthermore, the fixpoint query language $FO + LFP$ is more expressive than $Datalog^{\neg}$ [21]. Finally, $FO + LFP$ is less expressive than $QPTIME$ [10].

4 Weighted Query Languages

In this section, we compare the expressive power of a Boolean query language with the expressive power of a weighted query language where every query consists of vague search criteria. We will present a necessary and sufficient condition which characterizes the cases where the Boolean query language is less expressive than the weighted query language.

The Boolean query language and the weighted query language are of *different types*. The answer to a Boolean query is a set of data items, i.e. the set of items that satisfy a given Boolean condition. On the other hand, the answer to a weighted query is a function that assigns every data item a numerical value. This numerical value expresses the degree up to which the specified search criteria are satisfied.

Let $D = \{d_0, \ldots, d_{n-1}\}$ be the set of stored data items and let $\Phi = \{\varphi_0, \ldots, \varphi_{m-1}\}$ be a set of search criteria. Every search criterion φ_i is a function $\varphi_i : D \to [0,1]$, $d_j \mapsto \varphi_i(d_j)$. The value $\varphi_i(d_j)$ expresses how well the search criterion φ_i is satisfied by d_j. For instance, let φ_i be the search criterion $age(d_j) = 20$. The function φ_i can be modeled by means of a regression function which yields $\varphi_i(d_j) = 1.0$ if $age(d_j) = 20$. The value $\varphi_i(d_j)$ is decreasing when $|\, age(d_j) - 20 \,|$ is increasing [15].

For the subsequent analysis we have to know only how well the search criteria are satisfied by the data items. Hence, every data item d_j can be represented as a vector

$$d_j \quad = \quad (\varphi_0(d_j), \ldots, \varphi_{m-1}(d_j)). \tag{26}$$

We assume that the data items are uniquely identified by these vectors such that every instance of the database system can be represented by an $m \times n$ matrix A where

$$A_{i,j} \quad := \varphi_i(d_j). \tag{27}$$

According to our assumption, the data item is uniquely identified by the the j^{th} column of the matrix A. The *database system* $\mathcal{A}$ contains some matrices A, each representing an instance of the database system.

A *Boolean query* is a Boolean expression of search criteria. Thus, every search criterion φ_i is a Boolean query and $(q \wedge q')$, $(q \vee q')$, and $(\neg q)$ are Boolean queries if q and q' are Boolean queries. The set of all Boolean queries is denoted by QL_b.

The *answer* to a Boolean query consists of a set of data items. The power set of D is denoted by AS_b which represents the answer system. Thus, AS_b contains all possible answers to Boolean queries.

The Boolean query evaluation function $\alpha_b : \mathcal{A} \times QL_b \rightarrow AS_b$ is defined inductively.

$$\alpha_b(A, \varphi_i) := \{d_j \in D \mid A_{i,j} = 1\}) \quad (28)$$

$$\alpha_b(A, q \wedge q') := \alpha_b(A, q) \cap \alpha_b(A, q') \quad (29)$$

$$\alpha_b(A, q \vee q') := \alpha_b(A, q) \cup \alpha_b(A, q') \quad (30)$$

$$\alpha_b(A, \neg q) := D - \alpha_b(A, q) \quad (31)$$

The Boolean information system is represented by the following structure.

$$BIS = \langle \mathcal{A}, QL_b, AS_b, \alpha_b \rangle \quad (32)$$

As an example of a Boolean information system, we may think of a Boolean full-text retrieval system. Every search criterion φ_i corresponds to a keyword which either occurs in the document d_j or not. In the former case, $\varphi_i(d_j) = 1$ and in the latter case $\varphi_i(d_j) = 0$.

A *weighted query* is a vector $q = (q_0, \ldots, q_{m-1})^T$ where each component q_i is a weight expressing the importance of the search criterion φ_i. The weighted query language QL_w consists of all m-dimensional vectors, i.e. $QL_w = \mathrm{R}^m$.

The *answer* to a weighted query consists of a vector of so-called retrieval status values.

$$\alpha_w(A, q) := (RSV_w(q, d_0), \ldots, RSV_w(q, d_{n-1})) \quad (33)$$

The retrieval status value $RSV_w(q, d_j)$ expresses how well the search criteria specified by the query q are satisfied by the data item d_j. Such an answer can also be considered as a fuzzy set which specifies for every data item the probability that the item belongs to the answer set. As shown in [24], meaningful retrieval status values can be obtained by means of an appropriate positive definite $m \times m$ matrix M.

$$RSV_w(q, d_j) := \sum_{h,i} q_h M_{h,i} A_{i,j} \quad (34)$$

In matrix notation, equation (33) becomes $\alpha_w(A, q) = q^T MA$. In the simplest case, M is the identity matrix and the retrieval status value

is equal to the inner vector product of the query vector q and the item vector d_j. In this case, we have

$$RSV_w(q, d_j) \quad := \quad q_0 A_{0,j} + \ldots + q_{m-1} A_{m-1,j} \tag{35}$$

and equation (33) becomes $\alpha_w(A, q) = q^T A$. Equation 35 shows that the retrieval status value is maximal if all specified search criteria are satisfied. The objective of weighted retrieval is to achieve that the more the user knows about the desired data items the more likely he or she will find them. Unfortunately, Boolean information systems do not always have this behavior: The more the user knows about the desired data items and the more search criteria the user specifies, the more likely it happens that one of the specified search criteria is not completely correct. In this case, the desired data items are not retrieved by a Boolean information system. This problem is usually circumvented by artificial keys (e.g. employee numbers or social security numbers) such that the set of the desired data items can be specified by very few search criteria.

Let AS_w denote the set of possible answers to weighted queries. Thus, AS_w contains vectors consisting of n retrieval status values. We specify a weighted retrieval system in the following way.

$$WIS \quad = \quad \langle \mathcal{A}, QL_w, AS_w, \alpha_w \rangle \tag{36}$$

Our main result on the expressive power of Boolean retrieval and weighted retrieval will be based on the proposition given below. This proposition contains a necessary and sufficient condition which characterizes the cases where every Boolean query can be replaced by a weighted query. The proof of the proposition is given in Appendix A.

Proposition 6 *Let q_b be any Boolean query and let J_q be the set of indices of those items d_j that satisfy q_b, i.e. $J_q := \{j \mid d_j \in \alpha_b(A, q_b)\}$. There exists a weighted query q_w such that for all data items d_j in D,*

$$RSV_w(q_w, d_j) \quad = \begin{cases} 1.0 & \text{if } d_j \in \alpha_b(A, q_b) \\ 0.0 & \text{otherwise} \end{cases} \tag{37}$$

if and only if

$$j \in J_q \wedge d_j = \sum_{h \in J_q - \{j\}} c_h d_h \quad \Longrightarrow \quad \sum_{h \in J_q - \{j\}} c_h = 1, \tag{38}$$

$$j \notin J_q \wedge d_j = \sum_{h=0}^{n-1} c_h d_h \quad \Longrightarrow \quad \sum_{h \in J_q} c_h = 0. \tag{39}$$

The Boolean information system $BIS = \langle \mathcal{A}, QL_b, AS_b, \alpha_b \rangle$ and the weighted information system $WIS = \langle \mathcal{A}, QL_w, AS_w, \alpha_w \rangle$ are based on the same database system $\mathcal{A}$; however, their answer systems are of different types. Let A be a non-zero matrix representing a non-trivial instance of the database system. The Boolean queries generate a finite set of answers, i.e. $\{\alpha_b(A, q_b) \mid q_b \in QL_b\}$ is finite; however, the weighted queries generate an uncountable infinite set of answers, i.e. $\{\alpha_w(A, q_w) \mid q_w \in QL_w\}$ is uncountably infinite.

In what follows, we compare the expressive power of QL_b and the expressive power of QL_w. We define the correspondences between the answers of AS_b and the answers of AS_w as follows.

$$g : AS_b \rightarrow AS_w, D' \mapsto (\chi_{D'}(d_0), \ldots, \chi_{D'}(d_{n-1})) \tag{40}$$

The characteristic function $\chi_{D'} : D \rightarrow \mathsf{R}$ of the subset $D' \subseteq D$ is defined as usual.

$$\chi_{D'}(d_j) = \begin{cases} 1 & \text{if } d_j \in D' \\ 0 & \text{otherwise} \end{cases} \tag{41}$$

The correspondences between the instances of the database are given by the identity mapping $I_1 : \mathcal{A} \rightarrow \mathcal{A}, A \mapsto A$.

Proposition 7 *Let $BIS = \langle \mathcal{A}, QL_b, AS_b, \alpha_b \rangle$ be a Boolean Information System such that $range(\alpha_b)$ contains more than one element and let $WIS = \langle \mathcal{A}, QL_w, AS_w, \alpha_w \rangle$ be a Weighted Information System. Furthermore, let $I_1 : \mathcal{A} \rightarrow \mathcal{A}$ and $g : AS_b \rightarrow AS_w$ be the functions defined above. These functions determine the correspondences between the instances of the databases and the correspondences between the answers. The Weighted Query Language QL_w of WIS is more expressive than the Boolean Query Language QL_b of BIS with respect the correspondences given by I_1 and g iff all instances A of the database system $\mathcal{A}$ satisfy the conditions (38) and (39).*

Proof. Assume that all $A \in \mathcal{A}$ satisfy the conditions (38) and (39). $QL_b(BIS) \leq_{I_1,g} QL_w(WIS)$ follows immediately from Proposition 6. Because $range(\alpha_b)$ contains more than one element, there exists a query q_w and a document d_j such that $RSV_w(q_w, d_j) \neq 0$. Hence, q_w or $q_w + q_w$ do not have a corresponding query in QL_b because either $0 < RSV_w(q_w, d_j) \neq 1$ or $0 < RSV_w(q_w + q_w, d_j) \neq 1$. From (12) follows that QL_b and QL_w are not equally expressive.

Assume that $QL_b(BIS) <_{I_1,g} QL_w(WIS)$. By definition, for every Boolean query q_b there is a corresponding weighted query q_w such that $\forall A \in \mathcal{A} : g(\alpha_b(A, q_b)) = \alpha_w(A, q_w)$. From Proposition 6 follows that (38) and (39) are satisfied. $\Diamond$

The propositions presented in this section show that there are *redundancies* between Boolean query languages and weighted query languages. A necessary and sufficient condition was given which determines the cases where a Boolean query can always be replaced by an equivalent weighted query. In DB-theory, redundant data are related to inconsistent data [26, pp. 211]. From an algebraic point of view, the question arises whether redundancies between query languages are related to inconsistencies between query languages. Indeed, there are inconsistencies in the case of Boolean retrieval and weighted retrieval . The problem of these inconsistencies is the separability problem [8] which has been known in Information Retrieval for many years. Problems of the same type also occur in probabilistic databases [5] and in databases with maybe tuples [14]. Analogously to database design theory, we may argue that the separability problem would not occur if there were no redundancies between Boolean retrieval and weighted retrieval.

5 The Query Language FQL*

In this section, we introduce the Functional Query language FQL* as an example of a non-relational query language that provides a simple form of counting. Thus, as pointed out in Section 1, Chandra and Harel's notion of expressive power cannot be used to compare the expressive power of FQL* with the expressive power of relational query languages. In Section 6, we use our notion of expressive power to show that FQL* is more expressive than the fixpoint query languages.

We do not describe the full language FQL*. We restrict ourselves to those parts of FQL* which determine its expressive power. The syntax of FQL* is similar to the syntax of OSQL [6], [28]. Unlike the specifications of OSQL functions, the specifications of FQL* functions can be recursive or even mutually recursive. Like Datalog, FQL* has a fixpoint semantics that can be computed within finite time or more precisely, within a time bounded by a polynomial function of the size of the database. In this section, we give examples that show how the ancestor function and the bill of materials are formulated in FQL*.

Every FQL* function f is associated with a unique name denoted by $name(f)$. There are three different types of FQL* functions: stored functions, built-in functions, and derived functions. Similarly to OSQL [6], *stored functions* are stored by means of tables. For instance, a table may store the parents of persons. A *built-in function* is defined procedurally. For instance, given some numbers, the built-in function *sum* computes the sum of these numbers. Finally, *derived functions* are defined in a declarative way as described below.

FQL* functions are bag functions defined as follows. A *bag* A is a multiset which may contain an element y of a domain Y more than once [2]. In our case, a bag may even contain an element infinitely many times. The number of occurrences of an element y in a bag A is denoted by $A(y)$. Let $\overline{N} := \{0, 1, 2, \ldots, \omega\}$ be the set of natural numbers together with the limit ordinal ω denoting infinity. Every bag A is represented as a function $A : Y \to \overline{N}, y \mapsto A(y)$. The set of all bags with domain Y is denoted by $B(Y)$. A bag $A \in B(Y)$ is called a *finite bag* iff $\sum_{y \in Y} A(y) < \omega$. Subsequently, finite bags are written like sets, except that an element may occur several times (e.g. $A = \{a, a, b\}$). A bag $A \in B(Y)$ is contained in a bag $A' \in B(Y)$ iff every element $y \in Y$ occurs in A' at least as many times as in A.

$$A \sqsubseteq A' \quad :\Longleftrightarrow \quad \forall y : A(y) \leq A'(y) \tag{42}$$

Given a natural number n and two bags $A, A' \in B(Y)$, the intersection $A \sqcap A'$, the union $A \sqcup A'$, the sum $A + A'$, and the multiple $n * A$ are defined as follows.

$$A \sqcap A' : Y \to \overline{N}, y \mapsto min(A(y), A'(y)) \tag{43}$$

$$A \sqcup A' : Y \to \overline{N}, y \mapsto max(A(y), A'(y)) \tag{44}$$

$$A + A' : Y \to \overline{N}, y \mapsto A(y) + A'(y) \tag{45}$$

$$n * A : Y \to \overline{N}, y \mapsto n * A(y) \tag{46}$$

A *bag function* is a functional whose range is a set of bags. We will restrict ourselves to bag functions of the following types

$$f : X \to B(Y) \tag{47}$$

$$g : B(X) \to B(Y) \tag{48}$$

where X and Y are sets of objects or, in the terminology of object-oriented database systems, X and Y are classes. Every function value $f(x)$ is a bag, i.e. a mapping from Y to $\overline{N}$, and $f(x)(y)$ is the number

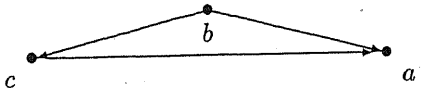

Figure 5: A directed graph.

of occurrences of y in the bag $f(x)$. The following example shows two bag functions associated with a directed graph.

Example 3 *Let $G = \langle V, E\rangle$ be the graph with vertices V and edges $E \subseteq V \times V$ as shown in Figure 5. The successor function $s : V \to B(V), v \mapsto s(v)$ determines the bag of immediate successor vertices for every vertex.*

$$s(a) = \{\} \quad s(b) = \{a, c\} \quad s(c) = \{a\}$$

The bag function $p : V \to B(V), v \mapsto p(v)$ assigns every vertex v a bag $p(v)$ which contains a vertex v' as many times as there are different paths from v to v'.

$$p(a) = \{\} \quad p(b) = \{a, a, c\} \quad p(c) = \{a\}$$

If the graph G represents a "part-of" relation, $s(v)$ contains the immediate subparts of v and $p(v)$ contains all subparts of v, i.e. $p(v)(v')$ determines how many times the part v' is contained in part v.

Given a natural number n and two bag functions $f : X \to B(Y)$ and $g : X \to B(Y)$, the intersection $f \sqcap g$, the union $f \sqcup g$, the sum $f + g$, and the multiple $n * f$ are defined as follows.

$$f \sqcap g : X \to B(Y), x \mapsto f(x) \sqcap g(x) \tag{49}$$

$$f \sqcup g : X \to B(Y), x \mapsto f(x) \sqcup g(x) \tag{50}$$

$$f + g : X \to B(Y), x \mapsto f(x) + g(x) \tag{51}$$

$$n * f : X \to B(Y), x \mapsto n * f(x) \tag{52}$$

Given a set of bag functions mapping X to $B(Y)$, we define a partial ordering $\sqsubseteq$ on this set. Let $f : X \to B(Y)$ and $g : X \to B(Y)$ be such bag functions. Using (42), the relation $\sqsubseteq$ on bag functions is defined by means of the relation $\sqsubseteq$ on bags.

$$f \sqsubseteq g \quad :\Longleftrightarrow \quad \forall x \in X : f(x) \sqsubseteq g(x) \tag{53}$$

Table F		Table H		Table FH	
X	Y	Y	Z	X	Z
a	p	p	u	a	u
b	p	p	v	a	v
b	q	q	v	b	u
		$\vdots$	$\vdots$	b	v
		q	v	$\vdots$	$\vdots$
				b	v

Figure 6: The tables F, H, and FH represent the bag functions $f : X \to B(Y), x \mapsto f(x)$, $h : Y \to B(Z), y \mapsto h(y)$, and $h \diamond f : X \to B(Z), x \mapsto h(f(x))$ respectively.

FQL* uses a *generalized form of composition* of bag functions. The generalized form facilitates not only the composition of functions $f : X \to B(Y)$ and $g : B(Y) \to B(Z)$, but also the composition of functions $f : X \to B(Y)$ and $h : Y \to B(Z)$. Before defining $h(f(x))$ in terms of bags, we define it by means of relational algebra. Assume that the bag functions f and h are represented conceptually by the tables F and H (Figure 6). These are conceptual representations because, for instance, the table H contains the tuple (q, v) infinitely many times. See the proof of Proposition 9 for the tables that are actually used to compute the composition.

The tables F and H determine the bag functions f and h by

$$f : X \to B(Y), x \mapsto \Pi[Y](\sigma[X = x](F)) \tag{54}$$

$$h : Y \to B(Z), y \mapsto \Pi[Z](\sigma[Y = y](H)) \tag{55}$$

where σ is the conventional select operator and Π is a special project operator which *does not* remove duplicates. In this way, the bag function h assigns the element q the bag $h(q)$ which contains v infinitely many times, i.e. $h(q)(v) = \omega$. The composition $h(f(x))$ is based on the natural join of F and H (the join attribute is Y).

$$h(f(x)) \quad := \quad \Pi[Z](\sigma[X = x](F \bowtie H)) \tag{56}$$

The resulting function $h \diamond f : X \to B(Z), x \mapsto h(f(x))$ is represented by the table FH (Figure 6).

In what follows, we define the generalized composition in terms of bags. Let $f : X \to B(Y)$, $g : B(Y) \to B(Z)$, and $h : Y \to B(Z)$ be

three bag functions. We define the compositions $g \circ f$ and $h \diamond f$ as follows.

$$g \circ f : X \to B(Z), x \mapsto g(f(x)) \tag{57}$$

$$h \diamond f : X \to B(Z), x \mapsto \sum_y f(x)(y) * h(y) \tag{58}$$

Note that the function value $\sum_y f(x)(y) * h(y)$ is defined by (46) and (45). The bag $h \diamond f(x)$ is obtained by determining the bag $f(x)$ and then adding up the bags $h(y)$ for every occurrence of y in $f(x)$. Sometimes, the composition $h \diamond f$ is called *flattening* because the bag $f(x)$ is made flat and h is applied to every occurrence of an element y in $f(x)$. For instance, assume that $f(a) = \{b, b, c\}$. Then, h is applied to every occurrence of an element in $f(a)$ to obtain, say $h(b) = \{d, d\}$ for the first occurrence of b in $f(a)$, $h(b) = \{d, d\}$ for the second occurrence of b in $f(a)$, and $h(c) = \{d, e\}$ for the occurrence of c in $f(a)$. Adding up the h values yields $(h \diamond f)(a) = h(b) + h(b) + h(c) = \{d, d, d, d, d, e\}$. We will use the following notational convention.

$$f''(f'(x)) \quad := \quad \begin{cases} f'' \circ f'(x) & \text{if } range(f') \subseteq dom(f'') \\ f'' \diamond f'(x) & \text{if } range(f') \subseteq B(dom(f'')) \end{cases} \tag{59}$$

Example 4 *Let $A = \{1, 1, 8, 9\}$ be a bag of $B(int)$ and let inc and sum be two bag functions defined as follows.*

$$inc : int \to B(int),\ i \mapsto \{i + 1\}$$

$$sum : B(int) \to B(int),\ X \mapsto \{\textstyle\sum_i X(i) * i\}$$

According to the generalized composition (59), we obtain, for instance, $inc(7) = \{8\}$, $inc(A) = \{2, 2, 9, 10\}$, $sum(A) = \{19\}$, $sum(inc(A)) = \{23\}$, and $inc(sum(A)) = \{20\}$.

We use a slightly modified version of the OSQL syntax [6] to specify derived FQL* functions. A set of n derived FQL* functions $f_0, \ldots, f_{n-1}$ is specified by n create statements each of the following form where $0 \leq q_i \leq r_i \leq s_i$.

create derived function
$f_i : X(i,0) \times \ldots \times X(i, q_i - 1) \to B(Y(i,0) \times \ldots \times Y(i, p_i - 1))$
such that $f_i(x_0, \ldots, x_{q_i - 1})$ **entails** $\chi_i(x_{q_i}, \ldots, x_{r_i - 1})$
for each $(x_{q_i}, \ldots, x_{r_i}, \ldots, x_{s_i - 1})$ **in** $X(i, q_i) \times \ldots \times X(i, s_i - 1)$
where $\varphi_i(\overline{f})(x_0, \ldots, x_{q_i - 1}, x_{q_i}, \ldots, x_{r_i - 1}, x_{r_i}, \ldots, x_{s_i - 1})$;

The sets $X(i,j)$ and $Y(i,j)$ denote sets of objects. The keyword "entails" denotes the relation $\sqsupseteq$ and χ_i denotes a composition of functions which maps $X(i,q_i)\times\ldots\times X(i,r_i-1)$ to $B(Y(i,0)\times\ldots\times Y(i,p_i-1))$. The predicate $\varphi_i(\overline{f}) : X(i,0)\times\ldots\times X(i,s_i-1)\rightarrow\{false,true\}$ may depend on the functions $\overline{f}=(f_0,\ldots,f_{n-1})$. This predicate is a propositional expression (and, or, not) of atomic formulas of the form $(\chi(\overline{z})$ entails $\chi'(\overline{z}'))$, where χ and χ' are compositions of functions. We will use the following abbreviations.

$$\begin{aligned}(\chi(\overline{z}) \doteq \chi'(\overline{z}')) &:= (\chi(\overline{z}) \textbf{ entails } \chi'(\overline{z}'))\\ &\quad\ \textbf{and } (\chi'(\overline{z}') \textbf{ entails } \chi(\overline{z}))\\ (\overline{z}=\overline{z}') &:= (\{\overline{z}\}=\{\overline{z}'\})\\ (\chi(\overline{z}) \textbf{ contains } \overline{z}') &:= (\chi(\overline{z}) \textbf{ entails } \{\overline{z}'\})\end{aligned}$$

Definition 6 *The predicate φ_i is called* monotonic *iff*

$$\overline{f}\sqsubseteq\overline{f}' \implies (\varphi_i(\overline{f})\Rightarrow\varphi_i(\overline{f}')) \tag{60}$$

where $\overline{f}\sqsubseteq\overline{f}'$ denotes that $f_i\sqsubseteq f_i'$ for $0\le i<n$. A bag function g mapping either X to $B(Y)$ or $B(X)$ to $B(Y)$ is called monotonic *iff $A\sqsubseteq A'$ implies that $g(A)\sqsubseteq g(A')$ for all bags A and A' in $B(X)$.*

Proposition 8 *Every bag function $f : X\rightarrow B(Y)$ is monotonic.*

Proof. $A\sqsubseteq A'\Rightarrow\forall x : A(x)\le A'(x)\Rightarrow\forall x:\forall y : A(x)*f(x)(y)\le A'(x)*f(x)(y)\Rightarrow\forall y : f(A)(y)=\sum_x A(x)*f(x)(y)\le\sum_x A'(x)*f(x)(y)=f(A')(y)\Rightarrow f(A)\sqsubseteq f(A')$. $\diamondsuit$

From the proposition above follows that a non-monotonic function is of the type $g : B(X)\rightarrow B(Y)$. For instance, the function $sum : B(int)\rightarrow B(int)$ introduced in Example 4 is not monotonic because $\{1,2\}\sqsubseteq\{1,2,3\}$ but $sum(\{1,2\})=\{3\}\not\sqsubseteq\{6\}=sum(\{1,2,3\})$. Before presenting the semantics of the create statements that specify derived functions, we give an example.

Example 5 *Assume that the function par : person $\rightarrow B(person)$ is a stored function which determines the parents for every person. The following create statement specifies the ancestor function.*

create derived function *anc* : *person* $\rightarrow B(person)$
such that $anc(x_0)$ **contains** x_1
for each x_1 **in** *person*
where $(par(x_0)\sqcup par(anc(x_0)))$ **contains** x_1);

The create statements specifying the derived functions $f_0, \ldots, f_{n-1}$ correspond to a system of n inequalities of the form

$$f_i(\overline{x}_i) \quad \supseteq \quad \chi_i(\pi_i(\sigma(\varphi_i(\overline{f}), \overline{x}_i)(X(i,0) \times \ldots \times X(i, s_i - 1)))). \quad (61)$$

where $\overline{x}_i = (x_0, \ldots, x_{q_i-1}) \in X(i,0) \times \ldots \times X(i, q_i - 1)$ and $\overline{f} = (f_0, \ldots, f_{n-1})$. The select operator $\sigma(\varphi_i(\overline{f}), \overline{x}_i)$ selects from $X(i,0) \times \ldots \times X(i, s_i - 1)$ those tuples $(x'_0, \ldots, x'_{s_i-1})$ for which $(x'_0, \ldots, x'_{q_i-1}) = \overline{x}_i$ and the predicate $\varphi_i(\overline{f})(x'_0, \ldots, x'_{s_i-1})$ is true. The set containing the selected tuples is considered as a bag of $B(X(i,0) \times \ldots \times X(i, s_i - 1))$. The function

$$\pi_i : X(i,0) \times \ldots \times X(i, s_i - 1) \rightarrow B(X(i, q_i) \times \ldots \times X(i, r_i - 1))$$

"projects" the tuples $(x'_0, \ldots, x'_{s_i-1})$ to $(x'_{q_i}, \ldots, x'_{r_i-1})$ without removing duplicates. In contrast to the function π_i, the function $\sigma(\varphi_i(\overline{f}), \overline{x}_i)$ cannot generate duplicates. The function

$$\chi_i : X(i, q_i) \times \ldots \times X(i, r_i - 1) \rightarrow B(Y(i,0) \times \ldots \times Y(i, p_i - 1))$$

represents a composition of functions. In a special case, the composition χ_i may represent the identity where $\chi_i(x_{q_i}, \ldots, x_{r_i-1})$ is equal to $\{(x_{q_i}, \ldots, x_{r_i-1})\}$. According to Proposition 8, the functions π_i and χ_i are monotonic functions. The function $\sigma(\varphi_i(\overline{f}), \overline{x}_i)$ is monotonic if the predicate $\varphi_i(\overline{f})$ is monotonic.

The inequality (61) reveals a certain similarity with relational algebra which also contains select and project operators. As shown at the outset of this section, the composition of functions corresponds to the natural join. In addition, there exists a similarity between (61) and Datalog. When the predicates p_i are considered as relations P_i, every rule $p_0 \leftarrow p_1 \wedge \ldots \wedge p_k$ corresponds to an inequality $P_0 \supseteq P_1 \cap \ldots \cap P_k$.

The create statements specifying the derived functions $f_0, \ldots, f_{n-1}$ correspond to n inequalities of the form (61). The *semantics* of these create statements is given by n bag functions representing a unique minimal solution of the n inequalities. A solution is called a *minimal solution* iff $\overline{f}' \not\sqsubseteq \overline{f}$ for every solution $\overline{f}'$ that is different from $\overline{f}$. The following proposition gives a sufficient stratification condition implying the existence of a unique minimal solution. Similarly to Datalog, the uniqueness of the solution is guaranteed by a bottom up computation described in the proof of the proposition which is given in Appendix A.

Proposition 9 *Assume that the functions $f_0, \ldots, f_{n-1}$ are specified by n create statements which correspond to a system of n inequalities of the form*

$$f_i(\overline{x}_i) \quad \sqsupseteq \quad \chi_i(\pi_i(\sigma(\varphi_i(\overline{f}), \overline{x}_i)(X(i,0) \times \ldots \times X(i, s_i - 1)))). \quad (62)$$

This system of n inequalities has a uniquely determined minimum solution if there exists a stratification function

$$\nu : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\},\ i \mapsto \nu(i) \qquad (63)$$

such that for every $i \in \{0, \ldots, n-1\}$, the following three conditions are satisfied.

1. *If f_j occurs in the specification of f_i, then $\nu(j) \leq \nu(i)$.*
2. *The domain $X(i,0) \times \ldots \times X(i, q_i - 1))$ of f_i is finite.*
3. *If f_j occurs in the specification of f_i and $\nu(j) = \nu(i)$, then φ_i is monotonic and $Y(i,0) \times \ldots \times Y(i, p_i - 1)$ is finite.*

The uniquely determined minimum solution is computable by a polynomial time algorithm, i.e. in time $O(P(|X|))$ where P denotes a polynomial and $|X|$ denotes the number of objects stored in the database.

The example given below contains a derived FQL* function that computes the bill of materials mentioned in the introduction. The *bill of materials* is the following graph problem. Given a graph, let $P(v)$ be the set of paths p starting at $v = s(p)$ and ending at $t(p)$. The source vertex $s(p)$ and the target vertex $t(p)$ may be identical and the path $p \in P(v)$ may contain cycles. If the cost of a single vertex v are denoted by $cost(v)$, the total cost of v are defined by

$$total\_cost(v) := \sum_{p \in P(v)} cost(t(p)).$$

The cost of every vertex is assumed to be non-negative. Thus, the total costs of certain vertices are infinite if the graph contains cycles. Even in the presence of cycles, the "bill of materials" as a graph problem is well defined and the derived functions given below provide the correct answer. According to Proposition 9, these functions are computed within polynomial time even though some function values may consist of infinite bags. Note that, in contrast to our approach, the approach described in [19] does not deal with cycles, i.e. all graphs are assumed to be acyclic.

Example 6 *Assume that* $subparts : parts \rightarrow B(parts)$ *is a stored function which determines for every part its immediate subparts. The stored function* $cost : parts \rightarrow B(int)$ *determines for every part the non-negative cost of this particular part without its subparts. The derived function* $all\_subparts$ *determines for every part* x *a bag of parts. This bag contains a part* y *as many times as* y *is contained in* x*.*

create derived function $all\_subparts : parts \rightarrow B(parts)$
such that $all\_subparts(x)$
entails $subparts(y) + subparts(all\_subparts(y))$
for each y **in** $parts$
where $(x = y)$;

In the case of the derived function $all\_subparts$*, a trivial combination of a selection function and of a "projection" function occurs because*

$$\pi(\sigma(true, x)(parts \times parts)) \quad = \quad \{x\}$$

In such cases, it is convenient to drop **"for each ... where ..."** *to obtain a more compact specification.*

create derived function $all\_subparts : parts \rightarrow B(parts)$
such that $all\_subparts(x)$
entails $subparts(x) + subparts(all\_subparts(x))$;

The derived function $total\_cost$ *determines, for every part, the cost for the part itself and of all of its subparts.*

create derived function $total\_cost : parts \rightarrow B(int)$
such that $total\_cost(x)$
entails $sum(cost(\{x\} + all\_subparts(x)))$;

We conclude this section by summarizing the main advantages of bag functions. First of all, there exists a fixpoint semantics for derived bag functions. Second, aggregate functions can be represented in a natural way, e.g. $sum(\{1,1\}) = \{2\}$. In relational query languages, more complicated operators (e.g. group by) are required to compute $1+1$ because the bag $\{1,1\}$ is not a relation. Third, bag functions provide a primitive form of counting. Fourth, the availability of infinite bags together with the capability of counting leads to an outstanding expressive power as will be shown in the next section. Finally, every FQL* function is computable in polynomial time. We believe that

database queries should be evaluable in a reasonable time and therefore, it should not be possible to formulate NP-complete problems such as the traveling salesman problem [16, pp. 211] or non-decidable problems such as Hilbert's tenth problem [13]. As mentioned in [4], a valuable query language is a tradeoff between expressive power and computational complexity.

6 The Expressive Power FQL*

In this section, we show that FQL* is more expressive than the fixpoint query languages. The fixpoint query languages have been introduced in Section 3. We first show that every fixpoint query can be expressed by means of FQL* functions. The correspondence between the predicates $R : D^a \rightarrow \{false, true\}$ and FQL* functions $f : D^a \rightarrow B(\{1\})$ is defined by

$$R \simeq f \quad :\Longleftrightarrow \quad \forall \overline{d} \in D^a : R(\overline{d}) \Leftrightarrow f(\overline{d}) \neq \{\}. \tag{64}$$

Let $f : D^a \rightarrow B(\{1\})$ be any FQL* function. We define the derived FQL* functions not_f and $exists_f$ as follows.

create derived function $not_f : D^a \rightarrow B(\{1\})$
  such that $not_f(d_0, \ldots, d_{a-1})$ **entails** y
  for each y **in** $\{1\}$
  where $f(d_0, \ldots, d_{a-1}) = \{\}$;

create derived function $exists_f : D^{a-1} \rightarrow B(\{1\})$
  such that $exists_f(d_1, \ldots, d_{a-1})$ **entails** y
  for each (d_0, y) **in** $D \times \{1\}$
  where $f(d_0, \ldots, d_{a-1})$ **contains** 1;

Subsequently, the formulas $R \vee S$ and $\forall x_0 : R(x_0, \ldots, x_{a-1})$ are regarded as abbreviations of $\neg(\neg R \wedge \neg S)$ and $\neg(\exists x_0 : \neg R(x_0, \ldots, x_{a-1}))$ respectively. From the following proposition it follows that every first-order query is expressible in FQL*.

Proposition 10

$$R \simeq f, S \simeq g \implies R \wedge S \simeq f \sqcap g \tag{65}$$

$$R \simeq f \implies \neg R \simeq not_f \tag{66}$$

$$R \simeq f \implies \exists x_0 : R(x_0, \ldots, x_{a-1}) \simeq exists_f \tag{67}$$

Proof. The first implication is proven as follows.

$$\begin{aligned} R \simeq f, S \simeq g &\implies R(\overline{d}) \Leftrightarrow f(\overline{d}) \sqsupseteq \{1\}, S(\overline{d}) \Leftrightarrow g(\overline{d}) \sqsupseteq \{1\} \\ &\implies R(\overline{d}) \wedge S(\overline{d}) \Leftrightarrow f(\overline{d}) \sqcap g(\overline{d}) \sqsupseteq \{1\} \\ &\implies R \wedge S \simeq f \sqcap g \end{aligned}$$

The remaining two implications are proven analogously. $\diamondsuit$

Let $\varphi(P, \overline{x})$ be a first-order formula with a distinguished r-ary predicate symbol P and r free variables $\overline{x} = (x_0, \ldots, x_{r-1})$. In addition to P and $\overline{x}$, the formula $\varphi(P, \overline{x})$ may also contain $\wedge$, $\vee$, $\neg$, $\exists$, $\forall$, $R_0, \ldots, R_{k-1}$, and bound variables. The formula $\varphi(P, \overline{x})$ is assumed to contain only positive occurrences of P. As shown in Section 3, the operator τ given by $\tau(P) := \{\overline{d} \in D^r \mid \varphi(P, \overline{d})\}$ has a least fixpoint $lfp(\tau)$. In what follows, we show that the fixpoint query $lfp(\tau)$ can be expressed by derived FQL* functions.

Given the relations $\overline{R} = (R_0, \ldots, R_{k-1})$, let $\overline{g} = (g_0 \ldots, g_{k-1})$ be k stored functions such that $R_i \simeq g_i$ for $0 \leq i < k$. Furthermore, let $\varphi(P, \overline{x})$ be any first-order formula as described above. According to Proposition 10, there exist derived functions $f_1, \ldots, f_{n-1}$ such that

$$P \simeq f_0 \implies \{\overline{d} \in D^r \mid \varphi(P, \overline{d})\} \simeq f_1. \tag{68}$$

The $n-2$ derived functions $f_2, \ldots, f_{n-1}$ correspond to subformulas of $\varphi(P, \overline{x})$. The derived function f_0 specified by

create derived function $f_0 : D^r \to B(\{1\})$
such that $f_0(d_0, \ldots, d_{r-1})$ **entails** $f_1(d_0, \ldots, d_{r-1})$;

corresponds to $lfp(\tau)$, i.e. $lfp(\tau) \simeq f_0$, where f_0 depends on the other derived functions $f_1, \ldots, f_{n-1}$.

According to [18], every fixpoint query can be expressed either by a first-order formula or by a formula containing one fixpoint operator and only positive occurrences of the distinguished predicate P. Hence, every fixpoint query can be expressed by derived FQL* functions whose specifications satisfy the three conditions of Proposition 9. On the other hand, not every derived FQL* function can be expressed as a fixpoint query. For instance, the derived function *total_cost* (Example 6) cannot be expressed by a fixpoint query as shown in the introduction (Figure 2). In the rest of this section, the new notion of expressive power is used to show that FQL* queries are more expressive than fixpoint queries.

We start with defining the information system IS_{FQL} which is accessible through FQL*.

$$IS_{FQL} = \langle DBS_{FQL}, QL_{FQL}, AS_{FQL}, \alpha_{FQL} \rangle \quad (69)$$

An instance $\overline{g} = (g_0, \ldots, g_{k-1})$ of the database system DBS_{FQL}, consists of k stored functions. The create statement

create stored function $g : X_0 \times \ldots \times X_{q-1} \rightarrow B(Y_0 \times \ldots \times Y_{p-1})$;

creates the stored function g which maps every tuple $(x_0, \ldots, x_{q-1})$ to the empty bag, i.e. for all $(x_0, \ldots, x_{q-1})$, $g(x_0, \ldots, x_{q-1}) = \{\}$. Non-empty function values are specified by means of insert statements.

insert $(y_0, \ldots, y_{p-1})$ **into** $g(x_0, \ldots, x_{q-1})$;

An FQL* query $\langle S, fnm \rangle$ consists of a set S and of a name fnm. The set S contains create statements specifying derived functions and the name fnm denotes a function. The evaluation function α_{FQL} is defined for the following arguments. Given an instance $\overline{g} = (g_0, \ldots, g_{k-1})$ and a query $q = \langle S, fnm \rangle$, the answer $\alpha_{FQL}(\overline{g}, q)$ is defined iff

1. fnm denotes a built-in function, or
2. fnm denotes one of the stored functions, i.e. $fnm = name(g_i)$, or
3. fnm denotes a derived function specified by S and S satisfies the three conditions of Proposition 9.

In the first case, the answer $\alpha_{FQL}(\overline{g}, q)$ is equal to the built-in function named fnm. This answer is independent of the stored functions and from the derived functions. In the second case, $\alpha_{FQL}(\overline{g}, q) = g_i$ if $fnm = name(g_i)$. In this case, the answer is independent of the derived functions. In the third case, $\alpha_{FQL}(\overline{g}, q) = f_i$ if $fnm = name(f_i)$ and the derived function f_i is determined by the unique minimum solution of the system of inequalities corresponding to S. The derived functions may depend on the stored functions as well as on the built-in functions.

The correspondences between the instances of the databases and the correspondences between the queries are defined as follows. Let

$$IS_{FO+LFP} = \langle DBS_{rel}, QL_{FO+LFP}, AS_{rel}, \alpha_{FO+LFP} \rangle \quad (70)$$

be the relational database system which is accessible through fixpoint queries as described in Section 3. The function

$$u : DBS_{rel} \rightarrow DBS_{FQL},\ (D, \overline{R}) \mapsto \overline{g} \tag{71}$$

assigns every instance $(D, \overline{R})$ a tuple of stored functions $u(D, \overline{R}) = (g_0, \ldots, g_{k-1})$ such that $R_i \simeq g_i$ for $0 \leq i < k$ where as usual, $\overline{R} = (R_0, \ldots, R_{k-1})$. The function

$$v : range(\alpha_{FO+LFP}) \rightarrow range(\alpha_{FQL}),\ R \mapsto f \tag{72}$$

assigns every answer $R \subseteq D^b$ a function $f : D^b \rightarrow B(\{1\})$ such that $R \simeq f$. With respect to the correspondences given by u and v, we have shown that FQL* is more expressive than $FO + LFP$.

Proposition 11 *The query language QL_{FQL} of IS_{FQL} is more expressive than QL_{FO+LFP} of IS_{FO+LFP} with respect to the correspondences given by u and v, i.e.*

$$QL_{FO+LFP}(IS_{FO+LFP}) \ <_{u,v}\ QL_{FQL}(IS_{FQL}). \tag{73}$$

Proof. At the outset of this section, it was shown that for every fixpoint query $q \in QL_{FO+LFP}$, there exists a query $h(q) \in QL_{FQL}$ such that $\alpha_{FO+LFP}((D, \overline{R}), q) \simeq \alpha_{FQL}(u(D, \overline{R}), h(q))$. This implies that $QL_{FO+LFP}(IS_{FO+LFP}) \leq_{u,v} QL_{FQL}(IS_{FQL})$. Finally, from $QL_{FO+LFP}(IS_{FO+LFP}) =_{u,v} QL_{FQL}(IS_{FQL})$ would follow that the fixpoint queries were able to count. Since, this is not the case, QL_{FQL} is more expressive than QL_{FO+LFP}. $\Diamond$

7 Conclusions

It was argued that a new notion of expressive power is needed to compare non-relational query languages, particularly, advanced query languages with counting abilities. The proposed new notion of expressive power has been shown to facilitate the comparison of the expressive power of such query languages. When restricted to relational query languages, the new notion of expressive power is equivalent to the notion of expressive power by Chandra and Harel.

The new notion of expressive power has been serving as a platform to derive new results. First, the expressive power of a Boolean query language has been compared with the expressive power of a weighted

query language. Second, FQL* has been presented as an example of a non-relational query language with an outstanding expressive power. We described only those parts of the query language FQL* that are concerned with its expressive power. It was shown that the expressive power of FQL* surpasses the expressive power of fixpoint queries. This result is mainly due to the use of bag functions which encompass a primitive form of counting together with a primitive form of infinity. Thus, the complete lattice of bag functions may serve as an interesting alternative to the conventional power set of Herbrand interpretations.

A Theorems and Proofs

Proposition 6 *Let q_b be any Boolean query and let J_q be the set of indices of those items d_j that satisfy q_b, i.e. $J_q := \{j \mid d_j \in \alpha_b(A, q_b)\}$. There exists a weighted query q_w such that for all data items d_j in D,*

$$RSV_w(q_w, d_j) = \begin{cases} 1.0 & \text{if } d_j \in \alpha_b(A, q_b) \\ 0.0 & \text{otherwise} \end{cases} \tag{74}$$

if and only if

$$j \in J_q \wedge d_j = \sum_{h \in J_q - \{j\}} c_h d_h \implies \sum_{h \in J_q - \{j\}} c_h = 1, \tag{75}$$

$$j \notin J_q \wedge d_j = \sum_{h=0}^{n-1} c_h d_h \implies \sum_{h \in J_q} c_h = 0. \tag{76}$$

Proof. We define a Boolean retrieval function $RSV_b : QL_b \times D \to \mathsf{R}$ as follows.

$$RSV_b(q_b, d_j) := \begin{cases} 1.0 & \text{if } d_j \in \alpha_b(A, q_b) \\ 0.0 & \text{otherwise} \end{cases}$$

Then, condition (74) is equivalent to $RSV_b(q_b, d_j) = RSV_w(q_w, d_j)$ for all data items d_j in D.

First, we show how (75) and (76) imply the existence of a weighted query q_w such that for all $d_j \in D$, $RSV_b(q_b, d_j) = RSV_w(q_w, d_j)$. According to (26), every item d_j is regarded as a column vector of the feature-item matrix A. Let q_b be any Boolean query. In what follows, we show the existence of a corresponding query q_w. Without loss of generality, we assume that the items are numbered in such a way that

- the vectors $d_0, \ldots, d_{u-1}, d_u, \ldots, d_{u+v-1}$ are linearly independent,
- the items $d_0, \ldots, d_{u-1}$ satisfy the query q_b,
- the items $d_u, \ldots, d_{u+v-1}$ do not satisfy the query q_b,
- every vector d_j that satisfies the query q_b is a linear combination of the vectors $d_0, \ldots, d_{u-1}$, and
- every vector $d_j \in D$ is a linear combination of $d_0, \ldots, d_{u+v-1}$.

It is easy to show that the items can always be numbered in such a way.

Let A_{u+v} be the submatrix of A consisting of the first $u+v$ columns. Since the columns of A_{u+v} are linearly independent and M is positive definite, the columns of MA_{u+v} are also linearly independent. Thus, MA_{u+v} contains $u+v$ linearly independent rows.

Without loss of generality, we assume that the first $u+v$ rows of MA_{u+v} are linearly independent. The submatrix of MA_{u+v} consisting of the first $u+v$ rows is denoted by B. The matrix B is a regular $(u+v) \times (u+v)$ matrix which is also a submatrix of MA.

$$MA = \begin{array}{|c|c|} \hline B & \\ \hline & \\ \hline \end{array}$$

Since B is regular, the inverse matrix B^{-1} exists. The matrix C is obtained by adding $m-u-v$ zero vectors to the right of B^{-1}.

$$C := \begin{array}{|c|c|} \hline B^{-1} & 0 \\ \hline \end{array}$$

The $(u+v)$-dimensional row vector r^T consists of u ones and of v zeros.

$$r^T := (1, \ldots, 1, 0, \ldots, 0)$$

When multiplying C and MA, we obtain a $(u+v) \times n$ matrix where the left square corresponds to the unit matrix E.

$$CMA = \begin{array}{|c|c|} \hline E & * \\ \hline \end{array}$$

When multiplying r^T and CMA, we obtain an n-dimensional row vector where the first u components are equal to 1 and the next v components are equal to zero.

$$r^T CMA \quad = \quad (1,\ldots,1,0,\ldots,0,*,\ldots,*)$$

We define $q_w := C^T r$. To show that for all $d_j \in D$, $RSV_b(q_b, d_j) = RSV_w(q_w, d_j)$, we distinguish four cases. Let $d_j \in D$ be any item.

Case I: $j \in \{0,\ldots,u-1\}$. Since d_j is the j^{th} column of A, $RSV_w(q_w, d_j) = q_w^T M d_j = r^T CM d_j$ is equal to the j^{th} component of $r^T CMA$ which is equal to $1 = RSV_b(q_b, d_j)$.

Case II: $j \in \{u,\ldots,u+v-1\}$. Similarly to case I, $RSV_w(q_w, d_j)$ is equal to the j^{th} component of $r^T CMA$ which is equal to $0 = RSV_b(q_b, d_j)$.

Case III: $j \in \{u+v,\ldots,n-1\}$ and d_j satisfies q_b, i.e. $RSV_b(q_b, d_j) = 1$. Since d_j is not a base vector, $d_j = \sum_{h=0}^{u-1} c_h d_h$. From the linearity of RSV_w and from (75) follows $RSV_w(q_w, d_j) = \sum_{h=0}^{u-1} c_h RSV_w(q_w, d_h) = \sum_{h=0}^{u-1} c_h = 1 = RSV_b(q_b, d_j)$.

Case IV: $j \in \{u+v,\ldots,n-1\}$ and d_j does not satisfy q_b, i.e. $RSV_b(q_b, d_j) = 0$. Since d_j is not a base vector and $d_j = \sum_{h=0}^{u+v-1} c_h d_h$. From the linearity of RSV_w and finally, from (76) follow the equalities $RSV_w(q_w, d_j) = \sum_{h=0}^{u+v-1} c_h RSV_w(q_w, d_h) = \sum_{h=0}^{u-1} c_h 1 + \sum_{h=u}^{u+v-1} c_h 0 = 0 = RSV_b(q_b, d_j)$.

Second, we show that if (75) and (76) are not satisfied, the existence of a weighted query q_w such that $RSV_w(q_w, d_j) = RSV_b(q_b, d_j)$ yields a contradiction.

Assume that (75) is not satisfied, i.e. there exists a j such that $j \in J_q$, $d_j = \sum_{h \in J_q - \{j\}} c_h d_h$, and $\sum_{h \in J_q - \{j\}} c_h \neq 1$. From the linearity of RSV_w we get a contradiction: $1 = RSV_b(q_b, d_j) = RSV_w(q_w, d_j) = \sum_{h \in J_q - \{j\}} c_h RSV_w(q_w, d_h) = \sum_{h \in J_q - \{j\}} c_h \neq 1$.

Assume that (76) is not satisfied, i.e. there exists a j such that $j \notin J_q$, $d_j = \sum_{h=0}^{n-1} c_h d_h$, and $\sum_{h \in J_q} c_h \neq 0$. Again, from the linearity of RSV_w there follows a contradiction: $0 = RSV_b(q_b, d_j) = RSV_w(q_w, d_j) = \sum_{h=0}^{n-1} c_h RSV_w(q_w, d_h) = \sum_{h \in J_q} c_h * 1 + \sum_{h \notin J_q} c_h * 0 = \sum_{h \in J_q} c_h \neq 0$. $\Diamond$

Proposition 9 *Assume that the functions $f_0,\ldots,f_{n-1}$ are specified by n create statements which correspond to a system of n inequalities of the form*

$$f_i(\overline{x}_i) \quad \sqsupseteq \quad \chi_i(\pi_i(\sigma(\varphi_i(\overline{f}), \overline{x}_i)(X(i,0) \times \ldots \times X(i, s_i - 1)))). \quad (77)$$

This system of n inequalities has a uniquely determined minimum solution if there exists a stratification function

$$\nu : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\},\ i \mapsto \nu(i) \tag{78}$$

such that for every $i \in \{0, \ldots, n-1\}$, *the following three conditions are satisfied.*

1. *If* f_j *occurs in the specification of* f_i, *then* $\nu(j) \leq \nu(i)$.
2. *The domain* $X(i,0) \times \ldots \times X(i, q_i - 1))$ *of* f_i *is finite.*
3. *If* f_j *occurs in the specification of* f_i *and* $\nu(j) = \nu(i)$, *then* φ_i *is monotonic and* $Y(i,0) \times \ldots \times Y(i, p_i - 1)$ *is finite.*

The uniquely determined minimum solution is computable by a polynomial time algorithm, i.e. in time $O(P(|X|))$ *where* P *denotes a polynomial and* $|X|$ *denotes the number of objects stored in the database.*

Proof. The proof of this proposition is not particularly complicated even though it is several pages long. The semantics of FQL* is based on a non-Boolean distributive lattice whereas the semantics of conventional query languages is based on Boolean lattices, e.g. the set of Herbrand interpretations. Thus, we have to develop a new theoretical framework from scratch. For this reason the following proof is quite long.

We first introduce some notations which will be used subsequently. We will say that a function f_i or an inequality $f_i(\ldots) \sqsupseteq \chi_i(\ldots)$ belongs to the stratification level k iff $\nu(i) = k$. In what follows, the definitions of least solutions and minimum solutions are given. Let L be the set of n-tuples $\overline{f} = (f_0, \ldots, f_{n-1})$ where f_i is a function mapping $X(i,0) \times \ldots \times X(i, q_i - 1)$ to $B(Y(i,0) \times \ldots \times Y(i, p_i - 1))$. Using (53), we define $\overline{f} \sqsubseteq \overline{f}'$ iff $f_i \sqsubseteq f_i'$ for $0 \leq i < n$. The structure $\langle L, \sqsubseteq \rangle$ is a partial ordering because $\sqsubseteq$ is reflexive, antisymmetric, and transitive. The n-tuple $\overline{f}$ is called a *solution* iff $\overline{f}$ satisfies the specified n inequalities. The n-tuple $\overline{f}$ is called a *least solution* iff it is a solution and every solution $\overline{f}'$ entails $\overline{f}$, i.e. $\overline{f}' \sqsupseteq \overline{f}$. Let $\overline{f}$ and $\overline{f}'$ be least solutions. By definition, $\overline{f} \sqsubseteq \overline{f}'$ and $\overline{f}' \sqsubseteq \overline{f}$ imply $\overline{f} = \overline{f}'$ because $\sqsubseteq$ is antisymmetric. Hence, if there exists a least solution then it is unique. The n-tuple $\overline{f}$ is called a *minimum solution* iff there exists no solution $\overline{f}'$ which is less than $\overline{f}$, i.e. $\overline{f}' \not\sqsubset \overline{f}$ for every solution $\overline{f}'$. Every least solution is a minimum solution but not vice versa.

Next, we show that the inequalities of every stratification level k have a least solution if the functions of the lower stratification levels $0, \ldots, k-1$ are already computed. Note that the least solution of the inequalities of a stratification level determines the derived functions of this stratification level. Hence, computing the least solution of the inequalities of a stratification level is equivalent to computing the derived functions of this stratification level. The derived functions of the lower levels can be considered as stored functions. We assume without restricting the generality that there is only one stratification level containing n inequalities. In what follows, we show the existence of a least solution of these inequalities.

Assume that there exists an index $i \in \{0, \ldots, n-1\}$ such that φ_i is non-monotonic or $Y(i,0) \times \ldots \times Y(i,p_i-1)$ is infinite. From the third condition of the proposition follows that the specification of the function f_i does not depend on the functions $f_0, \ldots, f_{n-1}$. In particular, $\varphi_i(\overline{f})$ does not depend on $\overline{f}$ nor does χ_i depend on $\overline{f}$. Hence, the functions χ_i and π_i and the predicates $\varphi_i(\overline{f}, \overline{x}_i)$ are well defined. In this case, the least solution of the inequality $f_i(\ldots) \sqsupseteq \chi_i(\ldots)$ is obviously given by

$$f_i(\overline{x}_i) \quad := \quad \chi_i(\pi_i(\sigma(\varphi_i(\overline{f}), \overline{x}_i)(X(i,0) \times \ldots \times X(i,s_i-1))))$$

and the function f_i is considered as a stored function like the derived functions of the lower stratification levels. We proceed by restarting with a smaller set of functions, i.e. $\{f_0, \ldots, f_{i-1}, f_{i+1}, \ldots, f_{n-1}\}$. If there exists a $j \in \{0, \ldots, i-1, i+1, \ldots, n-1\}$ such that φ_j is non-monotonic, χ_j contains a non-monotonic non-derived function, or $Y(j,0) \times \ldots \times Y(j,p_j-1)$ is infinite, then f_j is also considered as a stored function. This procedure is repeated until the set of derived functions is empty or for each remaining f_k, φ_k is monotonic and $Y(k,0) \times \ldots \times Y(k,p_k-1)$ is finite.

Hence, we may assume that for every $f_i \in \{f_0, \ldots, f_{n-1}\}$, φ_i is monotonic and $Y(i,0) \times \ldots \times Y(i,p_i-1)$ is finite. We define a function

$$T : L \to L, \overline{f} \mapsto (T_0(\overline{f}), \ldots, T_{n-1}(\overline{f}))$$

where $T_i(\overline{f})$ and f_i are functions of the same type.

$$\begin{array}{rcl} T_i(\overline{f}) : X(i,0) \times \ldots X(i,q_i-1) & \to & B(Y(i,0) \times \ldots Y(i,p_i-1)), \\ \overline{x}_i & \mapsto & T_i(\overline{f})(\overline{x}_i) \end{array}$$

The function $T_i(\overline{f})$ is defined by

$$T_i(\overline{f})(\overline{x}_i) \quad := \quad \chi_i(\pi_i(\sigma(\varphi_i(\overline{f}, \overline{x}_i)(X(i,0) \times \ldots \times X(i,s_i-1))))).$$

Since φ_i is monotonic, $\overline{f} \sqsubseteq \overline{f}'$ implies that

$$\begin{aligned}&\sigma(\varphi_i(\overline{f}), \overline{x}_i)(X(i,0) \times \ldots X(i, s_i - 1)) \\ &\quad \sqsubseteq \quad \sigma(\varphi_i(\overline{f}'), \overline{x}_i)(X(i,0) \times \ldots X(i, s_i - 1)).\end{aligned}$$

Furthermore, π_i and χ_i are always monotonic. Thus, $\overline{f} \sqsubseteq \overline{f}'$ implies that $T_i(\overline{f}) \sqsubseteq T_i(\overline{f}')$ and hence, T is monotonic.

Let $\langle L, \sqsubseteq \rangle$ be the partial ordering introduced above. Given any subset $M \subseteq L$, we define $\overline{g} = (g_0, \ldots, g_{n-1}) \in L$ such that

$$g_i(\overline{x}_i) \quad := \bigsqcup_{(f_0 \ldots f_{n-1}) \in M} f_i(\overline{x}_i).$$

It is easy to show that $\overline{g}$ is the least upper bound of M, i.e. $\overline{g} = lub(M)$. The existence of the greatest lower bound $glb(M)$ for every $M \subseteq L$ is shown similarly. Thus, $\langle L, \sqsubseteq \rangle$ is a complete lattice and according to a theorem by Tarski [25], the monotonic function $T : L \to L$ has a least fixpoint called $lfp(T)$. Obviously, this least fixpoint is the least solution of the n inequalities. Remember that, for simplicity, we assumed that all inequalities belong to the same stratification level. This completes the proof for the existence of a least solution for every stratification level.

Subsequently, we consider the general case, where the derived functions $f_0, \ldots, f_{n-1}$ belong to different stratification level. We will show that the bottom-up computation that starts at the stratification level 0 and ends at the stratification level $n-1$ provides a unique minimum solution of the n inequalities. This bottom-up procedure provides a solution because all inequalities are satisfied. The solution thus obtained is a minimum solution because, otherwise, the subsolutions of the particular stratification levels were not the least solutions. The order in which the derived functions are computed depends on the stratification function. Thus, the minimum solution seems to depend on the particular stratification function. This is not the case and it can be proven in the same way as it is proven that the minimum Herbrand model of a stratified Datalog program does not depend on the stratification function [3, Theorem 11].

In what follows, we present an algorithm to determine the minimum solution. Again, we consider only one stratification level and we assume that all derived functions $f_0, \ldots, f_{n-1}$ belong to this level. Furthermore, to keep the notation simple, we assume that there is only

one class of objects called X and every derived function is of the form $f_i : X \to B(X)$.

An iteration procedure is used to compute the least solution. Every function f_i is assigned a table τ_i. At the outset, every table τ_i contains for every $x, y \in X$ an entry $(x, y, 0, 0)$. The tables are updated when the iteration step is in progress. For every iteration step s, the tables $\tau_0, \ldots, \tau_{n-1}$ determine an element $\overline{f}^{(s)} = (f_0^{(s)}, \ldots, f_{n-1}^{(s)}) \in L$ by

$$f_i^{(s)}(x)(y) = m \iff (x, y, m, k) \in \tau_i.$$

Given an entry $(x, y, m, k) \in \tau_i$, the number k refers to the last iteration step when this entry has been updated.

Assume that s iteration steps are already performed. The next iteration step, $s+1$, is performed as follows. We distinguish two cases. First, if s divides $n \mid X \mid^2$, we compute the graph $\langle V, E \rangle$ defined by $V := \{0, \ldots, n-1\} \times X \times X$ and

$$\begin{aligned} &((i, v, w), (j, x, y)) \in E \\ &\quad :\iff T_j(\mu(i, v, w)(\overline{f}^{(s)}))(x)(y) > T_j(\overline{f}^{(s)})(x)(y). \end{aligned}$$

The function $\mu(i, v, w) : L \to L, \overline{f} \mapsto \mu(i, v, w)(\overline{f})$ is defined by

$$\begin{aligned} \mu(i, v, w) &:= (\mu_0(i, v, w), \ldots, \mu_{n-1}(i, v, w)) \\ \mu_h(i, v, w)(\overline{f})(t)(u) &:= \begin{cases} f_i(v)(w) + 1 & \text{if } (i, v, w) = (h, t, u) \\ f_h(t)(u) & \text{otherwise} \end{cases} \end{aligned}$$

Note that $\mu(i, v, w)(\overline{f})$ is identical to $\overline{f}$ except that the bag $f_i(v)$ contains an additional element w. An edge from (i, v, w) to (j, x, y) represents the following *dependency* between f_i and f_j. If a future iteration step requires an increment of $f_i(v)(w)$ then $f_j(x)(y)$ has to be incremented as well. In other words, if in one of the subsequent iteration steps, w is added to $f_i(v)$, then at least one element y has to be added to $f_j(x)$.

Let us assume that s divides $n \mid X \mid^2$ and the graph $\langle V, E \rangle$ has been computed. Then, every entry (x, y, m, k) in the table τ_j is replaced by the entry $(x, y, \omega, s+1)$ if the vertex (j, x, y) belongs to a cycle and $T_j(\overline{f}^{(s)})(x)(y) > T_j(\overline{f}^{(s-1)})(x)(y)$. In this case, $\overline{f}^{(s-1)} \sqsubseteq lfp(T)$ implies that $f_j(x)(y) = \omega$ is a necessary condition for $\overline{f} = lfp(T)$. Hence, from $\overline{f}^{(s)} \sqsubseteq lfp(T)$ follows that $\overline{f}^{(s+1)} \sqsubseteq lfp(T)$.

In the second case, where s does not divide $n \mid X \mid^2$, an entry (x, y, m, k) in τ_j is replaced by the entry $(x, y, m', s+1)$ whenever $m' = T_j(\overline{f}^{(s)})(x)(y) > m$. Because T is monotonic, from $\overline{f}^{(s)} \sqsubseteq lfp(T)$ follows again that $\overline{f}^{(s+1)} \sqsubseteq lfp(T)$.

We have considered two different types of iteration steps depending on whether s divides $n \mid X \mid^2$ or not. In both cases,

$$\overline{f}^{(s)} \sqsubseteq lfp(T) \implies \overline{f}^{(s+1)} \sqsubseteq lfp(T).$$

Since $\overline{f}^{(0)} \sqsubseteq lfp(T)$, it follows that $\overline{f}^{(s)} \sqsubseteq lfp(T)$ for every iteration step s.

When no new entries are inserted, the iteration is terminated. If t is the last iteration step, $\overline{f}^{(t)}$ is a fixpoint. This implies $lfp(T) \sqsubseteq \overline{f}^{(t)}$. Together with $\overline{f}^{(t)} \sqsubseteq lfp(T)$ it follows that the iteration procedure provides the least fixpoint because $\overline{f}^{(t)} = lfp(T)$.

It remains to show that the iteration procedure is a polynomial time algorithm. Let us consider paths without cycles in the graph G. The longest path without cycles along which an increment can be propagated is $\mid V \mid -1$. Thus, if the iteration procedure is not terminated after $\mid V \mid = n \mid X \mid^2$ iteration steps, the graph G contains cycles. In this case, at least one entry (x, y, ω, s) is inserted into a table and this entry is no more changed in the subsequent iteration steps. Since the number of entries is equal to $\mid V \mid$, we have an upper bound $\mid V \mid^2 = n^2 \mid X \mid^4$. Hence, the iteration procedure is a polynomial time algorithm. $\diamondsuit$

References

[1] A. V. Aho and J. D. Ullman. Universality of Data Retrieval Langages. In *Symposium on Principles of Programming Languages*, pages 110–117, 1979.

[2] J. Albert. Algebraic Properties of Bag Data Types. In *International Conference on Very Large Data Bases*, 1991.

[3] K. R. Apt, H. A. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, Los Altos, California, 1988. Morgan Kaufmann Publishers.

[4] F. Baader. A Formal Definition for the Expressive Power of Knowledge Representation Languages. In L. C. Aiello, editor, *European Conference on Artificial Intelligence*, pages 53–58, London, 1990. Pitman.

[5] D. Barbara, H. Garcia-Molina, and D. Porter. A Probabilistic Relational Model. In *Advances in Database Technology—EDBT'90*, pages 60–74, Berlin, 1990. Springer-Verlag.

[6] D. Beech. A Foundation for Evolution from Relational to Object Databases. In *Advances in Database Technology—EDBT'88*, pages 251–270, Berlin, 1988. Springer-Verlag.

[7] G. Birkhoff and J. D. Lipson. Heterogeneous Algebras. *Journal of Combinatorial Theory*, 8:115–133, 1970.

[8] D.A. Buell and D.H. Kraft. A Model for a Weighted Retrieval System. *Journal of the ASIS*, 32(3):211–216, 1981.

[9] A. Chandra and D. Harel. Computable Queries for Relational Databases. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.

[10] A. Chandra and D. Harel. Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences*, 25(1):99–128, 1982.

[11] A. K. Chandra. Theory of Database Queries. In *Symposium on Principles of Database Systems (PODS)*, pages 1–9, 1988.

[12] E. F. Codd. Relational Completeness of Database Sublanguages. In Rustin, editor, *Database Systems*. Prentice-Hall, 1972.

[13] M. Davis. Unsolvable Problems. In J. Barwise, editor, *Handbook of Mathematical Logic*. North-Holland, 1977.

[14] L.G. DeMichiel. Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains. *IEEE Transactions on Knowledge and Data Engineering*, 1(4):485–493, 1989.

[15] N. Fuhr. A Probabilistic Framework for Vague Queries and Imprecise Information in Databases. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *International Conference on Very Large Data Bases*, pages 696–707, Los Altos, CA, 1990. Morgan Kaufman.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, NY, 1979.

[17] Y. Gurevich. Toward Logic Tailored for Computational Complexity. In *Computation and Proof Theory, Lecture Notes in Mathematics*, pages 175–216. Springer-Verlag, 1984.

[18] Y. Gurevich and S. Shela. Fixpoint Extensions of First-Order Logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[19] M. A. W. Houtsma, P. M. G. Apers, and S. Ceri. Complex Transitive Closure Queries on a Fragmental Graph. In *International Conference on Database Theory*, pages 470–484. Springer-Verlag, 1990.

[20] N. Immerman. Relational Queries Computable in Polynomial Time. *Information and Control*, 68:86–104, 1986.

[21] P. Kolaitis. The Expressive Power of Stratified Logic Programs. *Information and Control*, 90(1):50–66, 1991.

[22] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

[23] L. V. Saxton and V. V. Raghavan. Design of an Integrated Information Retrieval/Database Management System. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):210–219, 1990.

[24] P. Schäuble. On the Compatibility of Retrieval Functions, Preference Relations, and Document Descriptions. Technical Report 113, ETH Zurich, Department of Computer Science, 1989.

[25] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and Its Applications. *Pacific Journal of Mathematics*, 5:285-309, 1955.

[26] J.D. Ullman. *Principles of Database Systems.* Computer Science Press, Rockville, Maryland, second edition, 1982.

[27] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, London, second edition, 1979.

[28] K. Wilkinson, P. Lyngbæk, and W. Hasan. The Iris Architecture and Implementation. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):63–75, 1990.

[29] B. Wüthrich. *Large Deductive Databases with Constraints.* PhD thesis, Swiss Federal Institute of Technology, 1991. VdF-Verlag, Zürich.

Gelbe Berichte des Departements Informatik

155	D. Crippa	A Special Case of the Dynamization Problem for Least Cost Paths
156	R. Griesemer	On the Linearization of Graphs and Writing Symbol Files
	C. Pfister (ed.), B. Heeb, J. Templ	Oberon Technical Notes
157	T. Weibel, G. Gonnet	An Algebra of Properties
158	M. Scholl (ed.)	Grundlagen von Datenbanken (Kurfassungen des 3.GI-Workshops, Volkse, 21. - 24.5.91)
159	K. Gates	Using Inverse Iteration to Improve the Divide and Conquer Algorithm
160	H. Mössenböck	Differences between Oberon and Oberon-2 The programming Language Oberon-2 (vergriffen)
161	S. Lalis	XNet: Supporting Distributed Programming in the Oberon Environment (vergriffen)
162	G. Weikum, C. Hasse	Multi-Level Transaction Management for Complex Objects: Implementation, Performance, Parallelism
163	J. Nievergelt et al.	e**X**perimental geometr**Y** **Z**urich: Software for Geometric Computation
164	G.H. Gonnet, H. Straub	The Dynamic Programming Algorithm as a Finite Automation
165	L. Adams, P. Arbenz	Towards a Divide and Conquer Algorithm for the Real Nonsymmetric.Eigenvalue Problem
166	E. Margulis	N-Poisson Document Modelling Revisited
167	H.E. Meier	Schriftgestaltung mit Hilfe des Computers Typographische Grundregeln mit Gestaltungsbeispielen (neue erweiterte Auflage)
168	B. Heeb, I. Noack	Hardware Description of the Workstation Ceres-3
169	M. Bronstein	Formulas for Series Computations
170	P. Arbenz	Divide and Conquer Algorithms for the Bandsymmetric Eigenvalue Problem
171	K. Simon, P. Trunz	On Transitive Orientation
172	A. Rosenthal, Ch. Rich, M.H. Scholl	Reducing Duplicate Work in Relational Join(s): A Unified Approach