ISSN 0105-8517

Optimal Detection of Query Injectivity

Michael I. Schwartzbach
Kim S. Larsen

DAIMI PB – 311
April 1990

COMPUTER SCIENCE DEPARTMENT
AARHUS UNIVERSITY
Ny Munkegade, Building 540
DK-8000 Aarhus C, Denmark

Optimal Detection of Query Injectivity

Michael I. Schwartzbach[1]
Kim S. Larsen[2]

Computer Science Department
Aarhus University
Ny Munkegade
DK-8000 Århus C, Denmark

Abstract

Most unary relational database operators can be described through functions from tuples to tuples. Injectivity of the specified function ensures that no duplicates are created in the relational result. This normally reduces the complexity of the query from $O(r \log r)$ to $O(r)$, where r is the number of tuples in the argument relation.

We consider functions obtained as terms over a general signature. The semantic properties of the operators are specified by Horn clauses generalizing functional dependencies. Relative to such specifications, we present an optimal algorithm for detecting injectivity of unary queries. The complexity of this algorithm is linear in the size of the query.

It turns out that relational functional dependencies are very easily incorporated into this framework. As a further result, we provide a Horn clause characterization of the functional dependencies that can be propagated to the result relation.

1 Introduction

Most unary relational database operators can be expressed in terms of a function from tuples to tuples. The result relation is the union of the individual applications of this function to the tuples in the argument relation. Examples of such operators are **project**, **extend**, and **rename**. Since the resulting tuples need not be distinct, it is necessary to remove potential duplicates. If r is the number of tuples in the argument relation,

[1]Internet address: mis@daimi.dk
[2]Internet address: kslarsen@daimi.dk

then this costs $O(r \log r)$, which will often dominate the complexity of the query. If the tuple function is known to be *injective*, then no duplicates can occur. For example, **extend** and **rename** are injective, whereas **project** is not.

It is often very convenient to directly specify a query by its tuple function. If $\bar{x}$ denotes the argument tuple, then the result tuple could be specified by

$$[\alpha_1\colon\ f_1(\bar{x}),\ \alpha_2\colon\ f_2(\bar{x}), \ldots,\ \alpha_k\colon\ f_k(\bar{x})]$$

Here, the α_i's are the new attribute names and the f_i's are functions. This is a query style that is vaguely similar to QBE [Zlo77]: in more involved cases it is easier to directly specify what the resulting tuples should look like than to compose a relational expression to achieve this. Of course, any query language can benefit from our results if its unary queries can be interpreted in the above manner. We give an example where the argument relation has attributes [Sal, Bon] (for Salary and Bonus). Then the query

$$[\mathrm{Tax}\colon\ 0.28\,(\mathrm{Sal}+\mathrm{Bon}),\ \mathrm{Pct}\colon\ (\mathrm{Sal}+\mathrm{Bon})/\mathrm{Sal},\ \mathrm{Large}\colon\ \mathrm{Bon} > 500]$$

could compute information for taxation purposes.

A generalization of this technique that allows all standard relational operators to be expressed in a similar manner (together with an infinitude of variations) is presented in [LSS89]. In the general case, the argument relations are not simply decomposed into tuples; instead, they are factorized into a matrix-like structure of subrelations. In [LSS89] we demonstrate that a fixed, uniform implementation of this new operator allows all the usual operators to retain their optimal complexities—except for the case of injective unary queries, where an unnecessary sorting is performed. This connection provides further motivation for obtaining an algorithm that decides injectivity of arbitrary tuple functions.

2 Outline

In this section we outline our approach to detecting injectivity.

We allow the involved functions to be terms generated over an arbitrary signature. This is much more general than merely considering a fixed, predefined set.

To decide injectivity we need, of course, some semantic information about the operators in this signature. The obvious choice is to specify an equational theory on the terms. This has, however, several disadvantages.

Firstly, injectivity of an equationally defined function is in general undecidable. Consider a theory with signature $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{f\}$, $\Sigma_2 = \{K\}$, and $\Sigma_3 = \{S\}$. The equations are

$$\begin{array}{rclcrcl} Kxy & = & x & \qquad & f(Kxy) & = & Kxy \\ Sxyz & = & xz(yz) & & f(Sxyz) & = & Sxyz \\ fa & = & t_1 & & f(fx) & = & fx \\ fb & = & t_2 & & & & \end{array}$$

where t_1 and t_2 are arbitrary SK-terms. The function f is injective if and only if $t_1 \neq t_2$. This is clearly undecidable as the SK-calculus is Turing complete. A possible solution to this problem is to seek a conservative algorithm that wrongly deems some functions to be non-injective. This is not a satisfactory solution, since it would not be very transparent which injective functions could in fact be detected. We would much rather restrict the permitted equations to a (large) class for which injectivity is decidable.

Secondly, a particular operator may be very easy to implement and yet be very difficult to axiomatize. This would inflict an unreasonable extra burden on the programmer.

Thirdly, an equational theory is not very modular. If we add a new operator, then we may have to relate it to all the existing ones. We would like to specify information for each operator purely locally.

A different approach addresses all of these concerns. In reality, we are not interested in which values an operator computes. We only want to know the properties of injectivity that it satisfies. Looking to functional dependencies in relational databases, we find a useful technique.

The semantic constraints on an operator in the signature will be specified as a Horn clause program with integers as propositions. Clauses such as

$$1 \leftarrow 2 \qquad \text{and} \qquad 2 \leftarrow 1$$

state that the resulting value, together with the second argument, completely determines the first argument, and vice versa. Both are true of e.g. the + operation on integers. Such specifications are local to the operator, they are easy to construct and verify, and they will prove quite sufficient to obtain interesting injectivity results.

In section 3, this framework is described in detail. In section 4, we develop an optimal algorithm for detecting injectivity of functions described by terms, and we prove its correctness. The algorithm is very efficient: its complexity is linear in the size of the term, which in the database application is the size of the query. In section 5, we extend the algorithm to consider functional dependencies in the argument relation. As a by-product of the injectivity algorithm, we obtain information about the dependencies in the result relation; this leads us to a complete syntactic characterization of the valid functional dependencies in the result.

3 The Framework

This section contains a precise statement of the problem we must solve. We work in an algebraic framework; standard definitions may be found in great detail in e.g. [Wir89]. However, our application is slightly non-standard.

3.1 (Σ, Δ)-algebras

For this application the types of values are not important, so we will work with homogeneous algebras. We use a fixed set of *attribute names* A whose elements are $x_1, \ldots, x_n$.

Definition 3.1 A *signature* is a ranked set

$$\Sigma = \bigcup_{k \in I\!N} \Sigma_k$$

where Σ_k contains the operators of arity k, and $A \subseteq \Sigma_0$. □

Definition 3.2 The *terms* over the signature Σ, $Term(\Sigma)$, are defined inductively to be the least set such that

- $\sigma \in \Sigma_0 \Rightarrow \sigma \in Term(\Sigma)$

- $$\begin{array}{l} \sigma \in \Sigma_k,\ k > 0, \text{ and } t_1, \ldots, t_k \in \mathit{Term}(\Sigma) \\ \Downarrow \\ \sigma(t_1, \ldots, t_k) \in \mathit{Term}(\Sigma) \end{array}$$

Henceforth, a term is an element of $\mathit{Term}(\Sigma)$. $\square$

Such terms will be used to denote function expressions.

Definition 3.3 An *operator clause* is of the form $\lfloor d_0 \leftarrow d_1, \ldots, d_m \rfloor$, where $d_0, d_1, \ldots, d_m \in \mathbb{N} \backslash \{0\}$ and $m \in \mathbb{N}$. $\square$

Operator clauses will be associated operator symbols to specify their properties of injectivity. If σ is an operator symbol of arity k, then the clause $\lfloor d_0 \leftarrow d_1, \ldots, d_m \rfloor$ should be interpreted: "from the result of an application of σ to k arguments *and* the d_1th to the d_mth argument we can uniquely determine the d_0th argument". (The floor symbols $\lfloor$ and $\rfloor$ are only used as delimiters).

Definition 3.4 A pair (Σ, Δ) is a *specification* if Σ is a signature and Δ maps operators to sets of operator clauses such that

$$\sigma \in \Sigma_k \ \wedge \ \lfloor d_0 \leftarrow d_1, \ldots, d_m \rfloor \in \Delta(\sigma) \ \Rightarrow \ \{d_0, d_1, \ldots, d_m\} \subseteq \{1, \ldots, k\}$$

We will use a fixed specification (Σ, Δ) throughout the paper. $\square$

Definition 3.5 If D is a set, then $\bar{v}$ denotes the tuple $\langle v_1, \ldots, v_p \rangle$, where $v_1, \ldots, v_p \in D$. If $i \in \{1, \ldots, p\}$ then $\bar{v}.i$ denotes the value v_i. If $p = n$ (the cardinality of the fixed set of attribute names A) and $a \in A$, then $a = x_i$ for some $x_i \in A$ and we let $\bar{v}.a$ denote v_i. $\square$

Our models will be algebras over (Σ, Δ). Such an algebra should, of course, be faithful to the information in (Σ, Δ), i.e. for each operator symbol in Σ, we will have an operator in our algebra with the correct arity (as specified by Σ) such that the properties of injectivity promised in Δ are actually fulfilled.

Definition 3.6 M is a (Σ, Δ)-algebra if it provides a carrier domain $\textsc{dom}_M$ and for each $\sigma \in \Sigma_k \backslash A$ a function σ^M such that

- $\sigma^M\colon \text{DOM}_M^k \to \text{DOM}_M$
- for each $\lfloor d_0 \leftarrow d_1, \ldots, d_m \rfloor \in \Delta(\sigma)$ we have for all $\bar{v}, \bar{w} \in \text{DOM}_M^k$:

$$\begin{array}{rcl} & \langle \bar{v}.d_1, \ldots, \bar{v}.d_m, \sigma^M(\bar{v})\rangle = \langle \bar{w}.d_1, \ldots, \bar{w}.d_m, \sigma^M(\bar{w})\rangle & \\ \Downarrow & & \\ & \bar{v}.d_0 = \bar{w}.d_0 & \end{array}$$

Let $Alg(\Sigma, \Delta)$ be the set of all (Σ, Δ)-algebras. As the specification (Σ, Δ) is fixed, an *algebra*, henceforth, denotes a member of $Alg(\Sigma, \Delta)$. □

A term t can in a given algebra be interpreted as a function of arity n.

Definition 3.7 Let M be an algebra and t a term. We obtain the function $t^M\colon \text{DOM}_M^n \to \text{DOM}_M$ from t as follows:

- if $t = x_i \in A$, then $t^M(\bar{v}) = \bar{v}.i$
- if $t \in \Sigma_k \backslash A$, then $t = \sigma(t_1, \ldots, t_k)$ for some $\sigma \in \Sigma_k$ and
$$t^M(\bar{v}) = \sigma^M(t_1^M(\bar{v}), \ldots, t_k^M(\bar{v}))$$

This simply interprets t as a function of the n variables x_i. □

3.2 The Decision Problem

What we are really interested in is injectivity in the usual mathematical sense. That is, when is a term, interpreted as a function in a model, injective?

Definition 3.8 Let M be an algebra. A term t is called *semantically injective w.r.t. M* if $t^M\colon \text{DOM}_M^n \to \text{DOM}_M$ is an injective function in the usual mathematical sense, i.e.

$$\forall \bar{v}, \bar{w} \in \text{DOM}_M^n\colon\ \bar{v} \neq \bar{w} \Rightarrow t^M(\bar{v}) \neq t^M(\bar{w})$$

We will use $\text{SEM}(t)$ to denote that t is semantically injective w.r.t. any algebra. □

Section 4 will establish the following result.

Main Result Let t be a term. There is a linear-time algorithm to decide $\text{SEM}(t)$. □

4 The Solution

It is not at all obvious how to decide a "semantic" property like $\textsc{Sem}(t)$. Therefore, we find a more "syntactic" property to check instead. Of course, we then show that these two properties are equivalent.

As seen from definition 3.2, terms are built from attribute names and constants using operator symbols to combine terms. We want to obtain complete information (as a Horn clause program) as to which attribute names can be retrieved from a term. That is, if we know the value of $t^M(\bar{v})$, for some model M and values $\bar{v}$, which v_i's can we then retrieve? If all v_i's can be retrieved, then t^M has a *left inverse* and, hence, is injective.

Definition 4.1 If t is a term, then $\textsc{sub}(t)$ is the set of subterms of t defined recursively by

$$\textsc{sub}(t) = \{t\} \cup \bigcup_i \textsc{sub}(t_i)$$

where $t = \sigma(t_1, \ldots, t_k)$. □

Definition 4.2 Let t be a term. Then a t-clause is of the form

$$\lfloor s_0 \leftarrow s_1, \ldots, s_k \rfloor$$

where $s_i \in \textsc{sub}(t)$. □

Definition 4.3 The *denotation* $[\![t]\!]$ of a term t is the least finite set of t-clauses containing

- The main clause $\lfloor t \leftarrow \rfloor$.
- The clauses in $\tilde{\Delta}$, defined as

  $$\tilde{\Delta} = \bigcup_{s \in \textsc{sub}(t)} \bigcup_{\delta \in \Delta(\sigma)} \{\lfloor s_{d_0} \leftarrow s_{d_1}, \ldots, s_{d_m}, s \rfloor\}$$

  where $s = \sigma(s_1, \ldots, s_k)$ and $\delta = \lfloor d_0 \leftarrow d_1, \ldots, d_m \rfloor$.
- For each $\sigma \in \Sigma_k \backslash A$ and $\sigma(s_1, \ldots, s_k) \in \textsc{sub}(t)$, a functionality clause

  $$\lfloor \sigma(s_1, \ldots, s_k) \leftarrow s_1, \ldots, s_k \rfloor$$

As we shall later see, the denotation contains *all* the information pertinent to t. $\square$

The desired syntactic property can now be expressed by deductions in the denotation.

Definition 4.4 A term s can be *deduced* in $[\![t]\!]$, written $[\![t]\!] \vdash s$, if $\lfloor s \leftarrow s_1, \ldots, s_k \rfloor \in [\![t]\!]$ and $\forall i \in \{1, \ldots, k\}: \ [\![t]\!] \vdash s_i$. Deductions can conveniently be represented as *proofs* of the form

$$\dfrac{\dfrac{\vdots}{s_1} \cdots \dfrac{\vdots}{s_k}}{s}$$

i.e. finite trees where each line represents an application of a clause. $\square$

Definition 4.5 A term t is *syntactically injective*, written $\textsc{Syn}(t)$, if

$$\forall x_i \in A\colon \ [\![t]\!] \vdash x_i$$

As we shall see, this property exactly captures injectivity. $\square$

4.1 Soundness

We now set out to prove that syntactic injectivity implies semantic injectivity.

Definition 4.6 A t-clause $\lfloor s_0 \leftarrow s_1, \ldots, s_k \rfloor$ is *sound* if for all algebras M and $\bar{v}, \bar{w} \in \text{DOM}_M^n$ we have

$$\begin{array}{c} \langle s_1^M(\bar{v}), \ldots, s_k^M(\bar{v}), t^M(\bar{v})\rangle \ = \ \langle s_1^M(\bar{w}), \ldots, s_k^M(\bar{w}), t^M(\bar{w})\rangle \\ \Downarrow \\ s_0^M(\bar{v}) \ = \ s_0^M(\bar{w}) \end{array}$$

This is in line with the definition of operator clauses. $\square$

Lemma 4.7 All clauses in $[\![t]\!]$ are sound.

Proof For all algebras, M, we have

- Soundness of the main clause states that
$$\langle t^M(\bar{v})\rangle = \langle t^M(\bar{w})\rangle \Rightarrow t^M(\bar{v}) = t^M(\bar{w})$$
- Soundness of the $\tilde{\Delta}$-clauses states that
$$\begin{aligned} & \langle s^M_{d_1}(\bar{v}), \ldots, s^M_{d_m}(\bar{v}), s^M(\bar{v})t^M(\bar{v})\rangle \\ = \; & \langle s^M_{d_1}(\bar{w}), \ldots, s^M_{d_m}(\bar{w}), s^M(\bar{w})t^M(\bar{w})\rangle \end{aligned}$$
implies $s^M_{d_0}(\bar{v}) = s^M_{d_0}(\bar{w})$.
This holds since $s^M(\bar{u}) = \sigma^M(s^M_1(\bar{u}), \ldots, s^M_k(\bar{u}))$ and M is an algebra.
- The functionality clauses are sound since each σ^M is a function in the model.

□

We show that the deduced terms can be retrieved semantically.

Lemma 4.8 Let t and s be terms. Then for all algebras, M, we have
$$\begin{aligned} & [\![t]\!] \vdash s \\ & \Downarrow \\ & \forall \bar{v}, \bar{w} \in \mathrm{DOM}^n_M\colon\; t^M(\bar{v}) = t^M(\bar{w}) \Rightarrow s^M(\bar{v}) = s^M(\bar{w}) \end{aligned}$$

Proof We proceed by induction in the size of a proof of the form
$$\cfrac{\cfrac{\vdots}{s_1} \cdots \cfrac{\vdots}{s_k}}{s_0}$$

By hypothesis the result holds for the s_i's since they have shorter proofs. We have used the clause $\lfloor s_0 \leftarrow s_1, \ldots, s_k \rfloor$, which is already proved sound in lemma 4.7. But then
$$\begin{aligned} & t^M(\bar{v}) = t^M(\bar{w}) \\ & \Downarrow \\ & \forall i \in \{1, \ldots, k\}\colon\; s^M_i(\bar{v}) = s^M_i(\bar{w}), \text{ by the induction hypothesis} \\ & \Downarrow \\ & s^M_0(\bar{v}) = s^M_0(\bar{w}), \text{ by soundness of the clause} \end{aligned}$$

Notice that the base case is when s_0 is a fact. □

Theorem 4.9 (soundness) Let t be a term. If t is syntactically injective, then t is also semantically injective, i.e. $\textsc{Syn}(t) \Rightarrow \textsc{Sem}(t)$.

Proof

$$\begin{array}{ll}
& \textsc{Syn}(t) \\
\Downarrow & \\
& \forall x_i \in A\colon\ [\![t]\!] \vdash x_i, \text{ by definition} \\
\Downarrow & \\
& \forall M \in \mathit{Alg}(\Sigma, \Delta)\ \forall x_i \in A\ \forall \bar{v}, \bar{w} \in \textsc{dom}_M^n\colon \\
& \qquad\qquad t^M(\bar{v}) = t^M(\bar{w}) \Rightarrow \bar{v}.x_i = \bar{w}.x_i, \text{ by lemma 4.8} \\
\Downarrow & \\
& \forall M \in \mathit{Alg}(\Sigma, \Delta)\ \forall \bar{v}, \bar{w} \in \textsc{dom}_M^n\colon\ t^M(\bar{v}) = t^M(\bar{w}) \Rightarrow \bar{v} = \bar{w} \\
\Downarrow & \\
& \forall M \in \mathit{Alg}(\Sigma, \Delta)\ \forall \bar{v}, \bar{w} \in \textsc{dom}_M^n\colon\ \bar{v} \neq \bar{w} \Rightarrow t^M(\bar{v}) \neq t^M(\bar{w}) \\
\Downarrow & \\
& \textsc{Sem}(t), \text{ by definition}
\end{array}$$

□

4.2 Optimality

We now endeavor to prove that the other implication holds too, i.e. that semantic injectivity implies syntactic injectivity. Inspired by completeness proofs in logic, the most natural approach is to construct a falsifying model when syntactic injectivity fails. In our case, this means finding two different arguments which yield equal results. In order to obtain these, we introduce some distinct constants.

Definition 4.10 Let Σ^+ be Σ with for each $x_i \in A$ two extra constants $x_i^\circ, x_i^\bullet \in \Sigma_0^+$. If $t \in \mathit{Term}(\Sigma)$, then $t^\circ \in \mathit{Term}(\Sigma^+)$ is obtained by replacing each x_i with x_i°. We similarly define $t^\bullet$ by replacing each x_i with $x_i^\bullet$. □

We want to generate the term model from a very *weak* theory that is designed to be only just strong enough to prove the two results equal. We then show that this theory is too weak to force the arguments to be equal.

Definition 4.11 If $t \in \mathit{Term}(\Sigma)$, then $\textsc{mod}(t)$ is the initial model over Σ^+, where the defining equations are:

- The main equation: $t^\circ = t^\bullet$.
- The $\tilde{\Delta}$-equations: for each $\lfloor s_0 \leftarrow s_1, \ldots, s_k \rfloor \in \tilde{\Delta}$, we include

$$(s_1^\circ = s_1^\bullet) \wedge \ldots \wedge (s_k^\circ = s_k^\bullet) \Rightarrow s_0^\circ = s_0^\bullet$$

Values are congruence classes of $\mathit{Term}(\Sigma^+)$ under the least congruence generated by the defining equations. The class with representative s is denoted $[s]$. □

There is a standard theory of term models:

Lemma 4.12 $\textsc{mod}(t)$ is completely axiomatized by the theory $\textsc{eq}(t)$ obtained by adding to the defining equations:

- reflexivity: $a = a$
- symmetry: $a = b \Rightarrow b = a$
- transitivity: $(a = b) \wedge (b = c) \Rightarrow a = c$
- substitutivity:

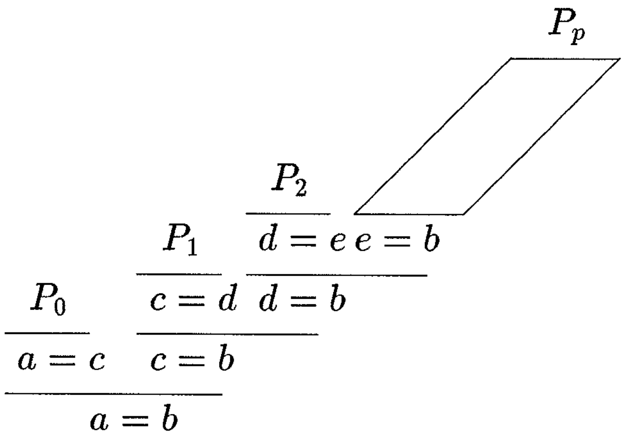
$$\begin{array}{c}(a_1 = b_1) \wedge \cdots \wedge (a_k = b_k)\\ \Downarrow \\ \sigma(a_1, \ldots, a_k) = \sigma(b_1, \ldots, b_k)\end{array}$$

i.e. in $\textsc{mod}(t)$ we have $[t_1] = [t_2]$ if and only if $\textsc{eq}(t) \vdash t_1 = t_2$.

Proof Immediate from definitions; for details see [Wir89]. □

The following result will facilitate the analysis of transitivity inferences in later proofs.

Lemma 4.13 Any proof in $\mathrm{EQ}(t)$ concluded with a transitivity inference has a normal form of the following kind:



where the concluding inferences in the P_i's are *not* transitivity. If such a concluding inference is symmetry, then the inference immediately above is neither symmetry nor transitivity. Let R_i be the last inference in P_i that is not symmetry. We call R_0 the *left-hand* inference and $R_1, \ldots, R_p$ the *right-hand* inferences.

Proof By applying the transformation,

$$\dfrac{\dfrac{b = c \qquad c = a}{b = a}}{a = b} \quad \longrightarrow \quad \dfrac{\dfrac{c = a}{a = c} \qquad \dfrac{b = c}{c = b}}{a = b}$$

we eliminate the symmetry inferences below transitivity inferences. Next, sequences of symmetry inferences are replaced by a single or none. Finally, by applying the transformation,

$$\dfrac{\dfrac{a = d \qquad d = c}{a = c} \qquad c = b}{a = b} \quad \longrightarrow \quad \dfrac{a = d \qquad \dfrac{d = c \qquad c = b}{d = b}}{a = b}$$

we move transitivity inferences to the right. □

We next show that we are justified in calling $\mathrm{MOD}(t)$ a *model.*

Lemma 4.14 $\mathrm{MOD}(t)$ is a (Σ, Δ)-algebra, when $\sigma \in \Sigma_k \backslash A$ is interpreted as the function $\sigma^{\mathrm{MOD}(t)}\colon ([r_1], \ldots, [r_k]) \mapsto [\sigma(r_1, \ldots, r_k)]$.

Proof Substitutivity ensures that $\sigma^{\text{MOD}(t)}$ is in fact a function. We must further show that it satisfies the requirements given by $\Delta(\sigma)$. Look at the definition of any such requirement. It is vacuously satisfied unless we have an equality of the form

$$\sigma^{\text{MOD}(t)}([a_1],\ldots,[a_k]) = \sigma^{\text{MOD}(t)}([b_1],\ldots,[b_k])$$

By lemma 4.12, we have such an equality if and only if

$$\text{EQ}(t) \vdash \sigma(a_1,\ldots,a_k) = \sigma(b_1,\ldots,b_k)$$

We proceed by induction in the size of such a proof and look at the last inference performed.

- $\tilde{\Delta}$-equation or the main equation: The conclusion is of the form $s^\circ = s^\bullet$, where $s \in \text{SUB}(t)$. By definition, all $\Delta(\sigma)$-requirements on such subterms are directly included as defining equations.
- Reflexivity: Here $a_i = b_i$ and we are done.
- Symmetry: The result follows trivially from the induction hypothesis.
- Transitivity: From lemma 4.13, we can assume that the proof is in normal form. We have two cases:
  - If the left-hand inference *or* all the right-hand inferences are substitutivity or reflexivity, then the following will be true of the final transitivity inference assumptions: one has all arguments pairwise equal, and the other satisfies by the induction hypothesis all the $\Delta(\sigma)$-clauses. It follows easily that the conclusion of the final transitivity inference also satisfies the $\Delta(\sigma)$-clauses.
  - If the left-hand inference *and* some right-hand inference both are $\tilde{\Delta}$-clauses or main clauses, then a shorter proof of the original equality can be found among the right-hand inferences. We now appeal to the induction hypothesis.
- Substitutivity: Here *all* the arguments must be pairwise equal, so $[a_i] = [b_i]$, and any $\Delta(\sigma)$-requirement is trivially satisfied.

□

The next result provides the important link between the model and the denotation.

Lemma 4.15 Model equality implies denotational deduction, i.e.

$$\forall s \in \textsc{sub}(t):\;\; \textsc{eq}(t) \vdash s^\circ = s^\bullet \;\Rightarrow\; [\![t]\!] \vdash s$$

Proof We inductively transform one proof into another. In $\textsc{eq}(t)$ we have a proof with conclusion $s^\circ = s^\bullet$. We consider the last inference performed:

- The main equation:

  $$\frac{}{t^\circ = t^\bullet}$$

  The result is trivial since $\lfloor t \leftarrow \rfloor \in [\![t]\!]$.

- A $\tilde{\Delta}$-equation:

  $$\frac{\dfrac{\vdots}{s_1^\circ = s_1^\bullet} \cdots \dfrac{\vdots}{s_k^\circ = s_k^\bullet}}{s^\circ = s^\bullet}$$

  By $s_i \in \textsc{sub}(t)$ and the induction hypothesis, we have $[\![t]\!] \vdash s_i$. The $\tilde{\Delta}$-equation comes from the Δ-clause $\lfloor s \leftarrow s_1, \ldots, s_k \rfloor$, so we conclude $[\![t]\!] \vdash s$.

- Reflexivity:

  $$\frac{}{s^\circ = s^\bullet}$$

  In this case s contains no x_i's. An easy induction shows that any such $s \in \textsc{sub}(t)$ can be derived from $[\![t]\!]$ using only functionality clauses.

- Symmetry: From lemma 4.13, we can assume that the symmetry inference is not immediately below a transitivity or another symmetry inference. Since we have a situation like

  $$\frac{\dfrac{\vdots}{s^\bullet = s^\circ}}{s^\circ = s^\bullet}$$

we can also eliminate $\tilde{\Delta}$-equations and the main equation. If symmetry is below reflexivity, then we proceed as above. Finally, if symmetry is below substitutivity, then we obtain shorter proofs of pairwise equality of the arguments in s° and $s^\bullet$. By the induction hypothesis, these can all be deduced, and with a functionality clause we deduce s.

- Transitivity: From lemma 4.13, we can assume that the proof is in normal form. If the left-hand or some right-hand inference is a $\tilde{\Delta}$-equation or the main equation, then we have a shorter proof of $s^\circ = s^\bullet$ and use the induction hypothesis. Otherwise, if all left-hand and right-hand inferences are reflexivity or substitutivity, then we obtain shorter proofs of pairwise equality of the arguments in s° and $s^\bullet$. By the induction hypothesis, these can all be deduced, and with a functionality clause we deduce s.

- Substitutivity:

$$\dfrac{\dfrac{\vdots}{s_1^\circ = s_1^\bullet} \cdots \dfrac{\vdots}{s_k^\circ = s_k^\bullet}}{s^\circ = s^\bullet}$$

  where $s^\circ = \sigma(s_1^\circ, \ldots, s_1^\circ)$ and $s^\bullet = \sigma(s_1^\bullet, \ldots, s_k^\bullet)$. Since $s \in \textsc{sub}(t)$ implies that $\textsc{sub}(s) \subseteq \textsc{sub}(t)$, we can apply the induction hypothesis to conclude that $[\![t]\!] \vdash s_i$. As we further have the functionality clause $\lfloor s \leftarrow s_1, \ldots, s_k \rfloor$, we are done.

Since we have covered all cases, the result follows. □

Theorem 4.16 (optimality) Let t be a term. If t is semantically injective, then t is also syntactically injective, i.e.

$$\textsc{Sem}(t) \;\Rightarrow\; \textsc{Syn}(t)$$

Proof Assume $\neg\textsc{Syn}(t)$. Then $\textsc{mod}(t)$ is a falsifying model:

$$\begin{array}{ll}
 & \neg\textsc{Syn}(t) \\
\Downarrow & \\
 & \exists x_i \in A\colon\ [\![t]\!] \nvdash x_i, \text{ by definition} \\
\Downarrow & \\
 & \exists x_i \in A\colon\ \textsc{eq}(t) \nvdash x_i^\circ = x_i^\bullet, \text{ by lemma 4.15} \\
\Downarrow & \\
 & \exists x_i \in A\colon\ [x_i^\circ] \neq [x_i^\bullet], \text{ by completeness of } \textsc{eq}(t) \\
\Downarrow & \\
 & \langle [x_1^\circ], \ldots, [x_n^\circ] \rangle \neq \langle [x_1^\bullet], \ldots, [x_n^\bullet] \rangle
\end{array}$$

But in $\textsc{mod}(t)$ we have $[t^\circ] = [t^\bullet]$, which is the same as

$$t^{\textsc{mod}(t)}(\langle [x_1^\circ], \ldots, [x_n^\circ] \rangle) = t^{\textsc{mod}(t)}(\langle [x_1^\bullet], \ldots, [x_n^\bullet] \rangle)$$

Hence, the interpretation of t is non-injective in $\textsc{mod}(t)$. The existence of such a model implies $\neg\textsc{Sem}(t)$. □

4.3 The Algorithm

We are interested in deciding $\textsc{Sem}(t)$, for a term t. But from theorem 4.9 and theorem 4.16 we can instead decide $\textsc{Syn}(t)$.

We first compute the denotation. It is a Horn clause program with subterms as propositions. By traversing the parse tree of t, we enumerate all such subterms and use these numbers for proposition symbols. We have $O(|t|)$ subterms of t.

Apart from the main clause, the denotation contains $\tilde{\Delta}$-clauses and functionality clauses. Let $|\sigma|$ denote the total number of propositions in $\Delta(\sigma)$. Then each subterm of t contributes at most $\max_{\sigma\in\Sigma} |\sigma|$ propositions to the $\tilde{\Delta}$-part of the denotation. As each subterm appears as a proposition at most twice in a functionality clause, this part of the denotation contains at most $2|t|$ propositions.

Thus, the denotation has size $O(|t| \max_{\sigma\in\Sigma} |\sigma|)$, and it can clearly be computed in this time, too. We are left with verifying the n deductions $[\![t]\!] \vdash x_i$. An algorithm in [DG84] can do this in a single computation in time $O(|[\![t]\!]| + n \log n)$. In conclusion, $\textsc{Syn}(t)$ can be decided in time

$$O(|t| \max_{\sigma\in\Sigma} |\sigma| + n \log n)$$

For the usual case of a fixed (Σ, Δ), this is the optimal time $O(|t|)$.

4.4 Databases Revisited

We are now able to (efficiently) decide if a term is injective, but the database application called for a *tuple* of terms $[t_1, t_2, \ldots, t_k]$. However, this is a term, too, if we regard $[\cdot, \cdot, \ldots, \cdot] \in \Sigma_k$ as a k-ary operator with $\Delta([\cdot, \cdot, \ldots, \cdot]) = \{\lfloor 1 \leftarrow \rfloor, \lfloor 2 \leftarrow \rfloor, \ldots, \lfloor k \leftarrow \rfloor\}$, i.e. a totally injective function. This also demonstrates a further generality of our result: it works equally well for relations where attributes are not atomic values, but themselves tuples or even more complicated structures.

To illustrate our technique we apply the algorithm to the example query given in the introduction:

$$[\text{Tax: } 0.28\,(\text{Sal} + \text{Bon}),\ \text{Pct: } (\text{Sal} + \text{Bon})/\text{Sal},\ \text{Large: } \text{Bon} > 500]$$

The available Δ-information is

$$\begin{aligned}
\Delta(+) &= \{\lfloor 1 \leftarrow 2 \rfloor, \lfloor 2 \leftarrow 1 \rfloor\} \\
\Delta(/) &= \{\lfloor 2 \leftarrow 1 \rfloor\} \\
\Delta(0.28) &= \{\lfloor 1 \leftarrow \rfloor\} \\
\Delta(>) &= \emptyset \\
\Delta([\cdot, \cdot, \cdot]) &= \{\lfloor 1 \leftarrow \rfloor, \lfloor 2 \leftarrow \rfloor, \lfloor 3 \leftarrow \rfloor\}
\end{aligned}$$

If Q denotes the entire query, then the denotation is as follows.

- Main clause:

  $\lfloor Q \leftarrow \rfloor$

- $\tilde{\Delta}$-clauses:

$\lfloor 0.28\,(\text{Sal} + \text{Bon}) \leftarrow Q \rfloor$	$\lfloor 1 \leftarrow \rfloor \in \Delta([\cdot, \cdot, \cdot])$
$\lfloor (\text{Sal} + \text{Bon})/\text{Sal} \leftarrow Q \rfloor$	$\lfloor 2 \leftarrow \rfloor \in \Delta([\cdot, \cdot, \cdot])$
$\lfloor \text{Bon} > 500 \leftarrow Q \rfloor$	$\lfloor 3 \leftarrow \rfloor \in \Delta([\cdot, \cdot, \cdot])$
$\lfloor \text{Sal} + \text{Bon} \leftarrow 0.28\,(\text{Sal} + \text{Bon}) \rfloor$	$\lfloor 1 \leftarrow \rfloor \in \Delta(0.28)$
$\lfloor \text{Sal} \leftarrow \text{Bon}, \text{Sal} + \text{Bon} \rfloor$	$\lfloor 1 \leftarrow 2 \rfloor \in \Delta(+)$
$\lfloor \text{Bon} \leftarrow \text{Sal}, \text{Sal} + \text{Bon} \rfloor$	$\lfloor 2 \leftarrow 1 \rfloor \in \Delta(+)$
$\lfloor \text{Sal} \leftarrow \text{Sal} + \text{Bon}, (\text{Sal} + \text{Bon})/\text{Sal} \rfloor$	$\lfloor 2 \leftarrow 1 \rfloor \in \Delta(/)$

- Functionality:

$$\begin{array}{l}\lfloor \mathrm{Sal}+\mathrm{Bon} \leftarrow \mathrm{Sal},\mathrm{Bon}\rfloor \\ \lfloor 0.28\,(\mathrm{Sal}+\mathrm{Bon}) \leftarrow (\mathrm{Sal}+\mathrm{Bon})\rfloor \\ \lfloor 500 \leftarrow \rfloor \\ \quad\vdots \end{array}$$

In this Horn clause program we have the following deductions:

$$\frac{\dfrac{\dfrac{\overline{Q}}{0.28\,(\mathrm{Sal}+\mathrm{Bon})}}{\mathrm{Sal}+\mathrm{Bon}} \qquad \dfrac{\overline{Q}}{(\mathrm{Sal}+\mathrm{Bon})/\mathrm{Sal}}}{\mathrm{Sal}}$$

$$\frac{\dfrac{\vdots}{\mathrm{Sal}} \qquad \dfrac{\vdots}{\mathrm{Sal}+\mathrm{Bon}}}{\mathrm{Bon}}$$

where the last deduction can be completed with subdeductions from the first. Hence, the query is injective.

5 Functional Dependencies

Our term clauses resemble generalized functional dependencies. We can exploit this connection in various ways.

In our framework a functional dependency is a term clause in which all terms are singleton attribute names; it is well-known that functional dependencies can be expressed as Horn clauses [Mai83].

Definition 5.1 If $\varphi = \lfloor x_{d_0} \leftarrow x_{d_1}, \ldots, x_{d_m}\rfloor$ is a functional dependency, then we define $\varphi(\bar{v},\bar{w})$ to denote that

$$\langle \bar{v}.d_1, \ldots, \bar{v}.d_m\rangle = \langle \bar{w}.d_1, \ldots, \bar{w}.d_m\rangle \;\Rightarrow\; \bar{v}.d_0 = \bar{w}.d_0$$

If Φ is a set of functional dependencies, then $\Phi(\bar{v},\bar{w})$ is an abbreviation for $\forall\varphi \in \Phi\colon\ \varphi(\bar{v},\bar{w})$. □

5.1 Incorporating Dependencies

When the argument relation satisfies a set of functional dependencies, Φ, then more queries will be injective. Accordingly, we generalize the definitions of semantic and syntactic injectivity.

Definition 5.2 We introduce $\textsc{Sem}_\Phi$ and $\textsc{Syn}_\Phi$ as

$$\begin{aligned} \textsc{Sem}_\Phi(t): \quad & \forall M \in Alg(\Sigma, \Delta)\, \forall \bar{v}, \bar{w} \in \textsc{dom}_M^n: \\ & \qquad \bar{v} \neq \bar{w} \wedge \Phi(\bar{v}, \bar{w}) \Rightarrow t^M(\bar{v}) \neq t^M(\bar{w}) \\ \textsc{Syn}_\Phi(t): \quad & \forall x_i \in A\colon\ [\![t]\!] \cup \Phi \vdash x_i \end{aligned}$$

Notice that $\textsc{Sem}_\emptyset = \textsc{Sem}$ and $\textsc{Syn}_\emptyset = \textsc{Syn}$. □

Theorem 5.3 Semantic and syntactic injectivity are equivalent, i.e.

$$\textsc{Sem}_\Phi \Leftrightarrow \textsc{Syn}_\Phi$$

Proof Straightforward generalization of the methods from section 4. □

Of course, the same linear-time algorithm can still be used to decide syntactic injectivity. This extension greatly increases the importance of the results obtained in the previous sections.

5.2 Propagating Dependencies

In a database model where functional dependencies are used, these must be specified manually each time a new relation is created. This is necessary because functional dependencies reflect the *semantic* view of the data and cannot be inferred from mere tuple values. However, if a relation is created by a unary query that has been analyzed using our technique, and attributes retain their meanings, then we can determine all the functional dependencies that will be valid for the new relation. For example, if Id is a key in the scheme [Id, Name, Age], then it will also be a key in the projection [Id, Name], provided both relations exist in the same semantic framework.

Definition 5.4 If t is a term, then the set of functional dependencies

$$\Psi = \{\psi \mid \forall M \in Alg(\Sigma, \Delta)\ \forall \bar{v}, \bar{w} \in \mathrm{DOM}_M^n\colon\ \psi(t^M(\bar{v}), t^M(\bar{w}))\}$$

is the *maximal* one satisfied by all pairs of tuples in all t-results. $\square$

This is another interesting semantic entity for which we can provide a syntactic characterization. As before, the query looks like $[t_1, t_2, \ldots, t_k]$. We express the new dependencies as Horn clauses with propositions taken from the set $\{t_1, t_2, \ldots, t_k\}$; of course, these can be rephrased in terms of the new attribute names.

Theorem 5.5 The dependencies in Ψ can be characterized as follows:

$$\lfloor t_{d_0} \leftarrow t_{d_1}, \ldots, t_{d_m} \rfloor \in \Psi$$
$$\Updownarrow$$
$$[\![t]\!] \cup \{\lfloor t_{d_1} \leftarrow \rfloor, \ldots, \lfloor t_{d_m} \leftarrow \rfloor\} \vdash t_{d_0}$$

Proof Another straightforward generalization of the methods from section 4. $\square$

This immediately gives rise to a naive exponential-time algorithm.

Incorporation and propagation of functional dependencies are orthogonal and can easily be combined.

6 Conclusion

We have shown how various semantic properties of queries can be captured syntactically. We feel that these results demonstrate how one can benefit from viewing relational operators as term-based expressions.

The concept of term clauses provides a basis for a new class of dependencies using term expressions rather than just single attributes. The consequences of this will be investigated further.

If unary queries are injective, then certain query rewrite rules will not increase efficiency. Hence, it may also be interesting to determine that queries are non-injective.

The optimization aspect of our results will be exploited in a new database language currently being implemented [LSS$^+$].

References

[DG84] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[LSS+] Kim S. Larsen, Erik M. Schmidt, Michael I. Schwartzbach, Erik Jacobsen, and Per O. Jensen. *RAS – a database language with functions.* In preparation.

[LSS89] Kim S. Larsen, Erik M. Schmidt, and Michael I. Schwartzbach. *A Universal Relational Operator.* PB 297, Computer Science Department, Aarhus University, 1989.

[Mai83] David Maier. *The Theory of Relational Databases.* Computer Science Press, Inc., 1983.

[Wir89] Martin Wirsing. *Algebraic Specification.* MIP 8914, Fakultät für Mathematik und Informatik, Universität Passau, June 1989. To appear in Handbook of Theoretical Computer Science from North-Holland.

[Zlo77] M. M. Zloof. Query-by-example: a data base language. *IBM Systems Journal*, 16(4):324–343, 1977.