Categorisation of Computable Database Queries

Mark Levene
University College London, University of London, Gower Street, London WC1E 6BT, U.K.
email: mlevene@cs.ucl.ac.uk

George Loizou
Birkbeck College, University of London, Malet Street, London WC1E 7HX, U.K.
email: george@dcs.bbk.ac.uk

Abstract

We present an alternative approach to that of Chandra and Harel [CHAN80] and Abiteboul and Vianu [ABIT90] in considering computable database queries, which are mappings from sets of records to sets of records. In particular, we view a computable database query as being realised via a Turing-computable mapping from strings to strings and an encoding, which encodes the input set of records into an appropriate string. An encoding of a set of records consists of two components: an ordering function, which orders the records in a set as well as the values of each record in the set, and an isomorphism, which maps the values in the records of the set to strings. An important class of encodings, called free encodings, whose isomorphism has the same semantics as the identity mapping on record values, is also defined.

Our analysis of computable database queries elucidates the notion of a computable database query by dealing with the problem of how a database language can be implemented on a standard Turing machine that does not cater directly for mappings from sets of records to sets of records. We carry out our analysis by categorising computable database queries into subclasses and by establishing the relationships that exist amongst these subclasses. We also investigate an equivalence relation on computable database queries; two computable database queries are related if they are realised via the same Turing-computable mapping, say δ. We prove the following interesting result regarding the cardinality of the equivalence class of a computable query with respect to the said equivalence relation: either δ does not realise any computable query, or δ realises exactly one computable query, or δ realises a countably infinite set of computable queries. Our final result shows that, by adding membership queries to the class of encoding-independent computable queries, the closure of the resulting extended class under composition of mappings is the set of all isomorphism-independent computable queries.

1. Introduction

There has been much interest in the notion of computable database queries (or simply computable queries) since the seminal work of [AHO79] and [CHAN80] and more recently [ABIT90], [ABIT91], [HULL93] and [SAZO93]. There are two fundamental differences between computable queries and Turing-computable mappings (also known as *partially recursive functions* [ROGE67]). The first difference is that Turing-computable mappings are mappings from strings to strings (or equivalently from natural numbers to natural numbers) whilst computable queries are mappings from finite sets of objects to finite sets of objects (cf. [GAND80]). The second difference is that the objects of computable queries are not strings or numbers but abstract objects (cf. [FRIE71]), in our case records (cf. [CARD91]), which are defined as total mappings from a finite set of attributes to a set of atomic values.

The aim of our work is to analyse the relationship between Turing-computable mappings and computable queries. Thus we define the semantics of a computable query as a Turing-computable mapping together with an encoding from finite sets of records (or simply sets) to strings. Consequently, we can evaluate a computable query by encoding the input set into a string, then use this string as an input to the given Turing-computable mapping, and finally decode the resulting output string to obtain the output set, which yields the result of the computable query. This definition of a computable query differs from the standard definition given in [CHAN80] and [ABIT90], wherein a computable query is defined directly as a computable mapping from sets to sets. Our analysis clarifies the notion of a computable query by dealing with the problem of how a database language can be implemented on a standard Turing machine that does not cater directly for mappings from sets to sets. We carry out our analysis by investigating subclasses of computable queries and by establishing the relationships that exist amongst these subclasses. (Throughout the paper, whenever no ambiguity arises, we will use the terms *class* and *subclass* to be synonymous with the terms set and subset, respectively; in addition, we will say that two classes or subclasses are *equivalent* if they are equal.)

In Section 2 we investigate in depth the semantics of encodings, an area which has been hitherto neglected in the database literature, and then formally define a computable query by making use of encodings. Informally, an encoding of a set consists of two components: an ordering function, which orders the records in a set as well as the values of each record in the set, and an isomorphism, which maps the values in the records of the set to strings. A decoding, which is the inverse of an encoding, also has two components: the inverse of the said isomorphism and a mapping which forgets the order imposed by the ordering function used in the encoding. Since, in general, there may not be an algorithm to convert any two encodings into each other we restrict the set of encodings to a set of *mutually convertible* encodings, which is a set of encodings that are algorithmically convertible to a given standard encoding [GARE79]. It follows that our (restricted set of) encodings are "reasonable" in the sense of [GARE79]. An important class of encodings, called *free encodings*, whose isomorphism maps record values to a corresponding natural string representation, is also defined. In a free encoding the isomorphism has the same semantics as the identity mapping on record values. We then define a computable query, τ, as a mapping from sets to sets which is computed via a Turing-computable mapping δ, an encoding ρ, which encodes the input to τ into a string for δ, and its inverse, the decoding ρ^{-1}, which decodes the output string of δ. As an abbreviation we say that τ is *realised* via δ and ρ.

In Section 3 we show the equivalence between the class of *generic computable queries* [CHAN80] and a subclass of our computable queries, namely the encoding-independent computable queries. Encoding-independent computable queries are queries which are independent of the two components of an encoding, that is they are independent of both the ordering function and the isomorphism which together make up the encoding (in other words, they are both order-independent and isomorphism-independent).

In Section 4 we investigate a subclass of our computable queries, namely the encoding-independent free-encoding computable queries. Computable queries in this subclass are encoding-independent in the sense that any free encoding can be used in the evaluation of a computable query without affecting the result of the computation. We show that the class of encoding-independent free-encoding computable queries is equivalent to the class of order-independent computable queries. Our main result shows the equivalence between the class containing the set of all C-generic computable queries for all finite sets C of distinguished constants [ABIT90] and the class of encoding-independent free-encoding computable queries. This result is obtained by exhibiting a *complete lattice* [DAVE90], partially ordered by set inclusion. This complete lattice comprises sets of computable queries together with the set of encoding-independent free-encoding computable queries; each of these sets is the set of C-generic computable queries, for some finite set C of distinguished constants. The set of encoding-independent free-encoding computable queries is shown to be the top element of this lattice and the set of encoding-independent computable queries is shown to be the bottom element of this lattice. The significance of this result is that we conclude that in order to show that a database language expresses the class of all C-generic computable queries we need only consider encoding-independent free-encoding computable queries.

In Section 5 we investigate an equivalence relation on the set of computable queries; two such queries are related if they are realised via the same Turing-computable mapping, say δ. In particular, we are interested in the cardinality of the equivalence class of a computable query, realised via δ and some encoding, with respect to the said equivalence relation; this equivalence class and its cardinality are denoted by Δ_δ and $|\Delta_\delta|$, respectively. In the case when $|\Delta_\delta| = 0$, there is no computable query τ such that τ is realised via δ and some encoding. On the other hand, when $|\Delta_\delta| = 1$ and τ is realised via δ and all possible encodings, then τ corresponds to an encoding-independent computable query. We show that both of the decision problems: is $|\Delta_\delta| = 0$? and is $|\Delta_\delta| = 1$? are undecidable. We also show the interesting result that if $|\Delta_\delta| > 1$, then $|\Delta_\delta| = \omega$. Our interpretation of this result is that it is not possible to obtain a finer partition of the class of computable queries with respect to the Turing-computable mappings that realise them, without putting restrictions on the set of encodings (for example, by only allowing free encodings).

In Section 6 we investigate enumerable database queries (or simply enumerable queries). We define an enumerable query R to be a (binary) relation, over sets, that can be enumerated via a Turing-enumerable relation (also known as a *recursively enumerable relation* [ROGE67]) T, after the ordered pairs in R have been encoded using an encoding ρ. We show the equivalence between the class of *generic enumerable queries* [ABIT90] and a subclass of enumerable queries, namely the class of encoding-independent enumerable queries. We show that all encoding-independent computable queries are encoding-independent enumerable queries but that there exists an encoding-independent enumerable query that has no corresponding encoding-independent computable query.

In Section 7 we define a new class of computable queries, namely the arbitrary-encoding computable queries, which are computable queries obtained by fixing the encoding used to realise a given computable query. For example, the fixed encoding may correspond to the physical layout of the database records. In fact we show that by fixing the encoding we do not incur any loss of computational expressiveness, since the class of arbitrary-encoding computable queries is equivalent to the class of computable queries. By adding membership queries (which are shown to be arbitrary-encoding computable queries that have the property of being order-dependent) to the class of encoding-independent computable queries, the closure of the resulting extended class under composition of mappings is the set of all isomorphism-independent computable queries. Finally, in Section 8 we give our concluding remarks.

2. Encodings and Computable Queries

In this section we investigate in depth the semantics of encodings, an area which has so far been neglected in the database literature, and then formalise the notion of a computable query. Prior to that we present the notation used throughout the paper and formalise the notions of a record and a set of records.

2.1. Notation

If S is a subset of T we write $S \subseteq T$ and if S is a proper subset of T we write $S \subset T$. We let ω denote the set of all natural numbers and let $|S|$ denote the cardinality of a countable set S. We use "o" to denote composition of mappings and let graph(f) denote the graph of a mapping f, where $(a, b) \in$ graph(f) if and only if f(a) is defined and f(a) = b. Unless otherwise stated we abbreviate the term *total ordering* of a set, say S, to *ordering* of S. Finally, we use the two index sets, $I = \{1,...,n\}$ and $J = \{1,...,m\}$, where $n, m \in \omega$.

2.2. Preliminary Definitions

Let Σ be a finite non-empty alphabet and Σ^* be the countably infinite set of all finite strings over Σ. We let TC denote the set of all Turing-computable mappings (also known as *partially recursive functions* [ROGE67]) from Σ^* to Σ^* and let TE denote the set of all Turing-enumerable (binary) relations (also known as *recursively enumerable relations* [ROGE67]) in Σ^*.

Next, let D be a countably infinite set of atomic values (or simply values) and let U be a countably infinite set of attributes.

Now, let $X \subset U$ be a finite set of attributes. An X-tuple (or simply a *record*, cf. [CARD91], whenever X is understood from context) is defined to be a total mapping t from X into D, such that $\forall A \in X, t(A) \in D$; in case $X = \emptyset$, we take t to be the $\emptyset$-tuple (or the *empty record*), denoted by $\langle\rangle$. We next take DOM(X) to be the set of all possible X-tuples (note that DOM(X) is countably infinite, since it is isomorphic to $D^{|X|}$).

Let $X = \{A_1, ..., A_m\}$. Then we denote an X-tuple t, where $\forall\, j \in J$, $t(A_j) = v_j$, by $<A_1 : v_1, ..., A_m : v_m>$. We note that $<A_1 : v_1, ..., A_m : v_m>$ is actually a shorthand for the set of ordered pairs $\{(A_1, v_1), ..., (A_m, v_m)\}$ representing the graph of the mapping t, since we have not assumed any underlying ordering of U.

Let $Dom = \cup \{DOM(X) \mid X \in P(U)\}$, where *P* denotes the finite powerset operator. Finally, let Ψ denote the set *P*(Dom) (note that Ψ is countably infinite, since all the sets in *P*(Dom) are finite and Dom is countably infinite) and let Φ denote the set of all mappings from Ψ to Ψ.

We assume that each record in Dom denotes some real world object. Therefore, a set in Ψ can be viewed as a database. For the purpose of this paper Dom can be viewed as a countably infinite set of first normal form records, where each record $t \in$ Dom is over some relation scheme [ULLM88], i.e. over a set of attributes $X \subset U$. That is, we consider the sets in Ψ to be *variable length relations* (or simply variable relations) [LEVE95]. Thus in the sequel, whenever no ambiguity arises, we use the terms set and variable relation interchangeably. We observe that although it is customary to define a relational database as a set of (fixed length) relations over a set of relation schemes [ULLM88], there is no loss of generality in our approach. This is due to the fact that we can easily partition a variable relation into a set of relations, say $\{r_1, ..., r_n\}$, over a set of relation schemes, say $\{X_1, ..., X_n\}$, such that $\forall\, i \in I$, $t \in r_i$ if and only if t is an X_i-tuple.

The following definition will be useful later on. The *active domain* of a set $S \in \Psi$, denoted by adom(S), is the set of attributes and atomic values occurring in the records of S, i.e.

$$adom(S) = \{v \mid \exists\, t \in S \text{ and } \exists\, A \in U \text{ such that } t(A) = v\} \cup \{A \mid \exists\, t \in S \text{ and } \exists\, v \in D \text{ such that } t(A) = v\}.$$

Example 2.1. In Figure 2.1 we show a linear representation of a variable relation which we call EMP. The semantics of EMP are: an employee has a NAME, earns a SALary, works in one DEPArTment and may have at most one SPouSe. In addition, a DEPT has one ManaGeR and a MGR has one SECretary. An equivalent tabular representation of EMP is shown in Figure 2.2.

{<DEPT : Computing,	NAME : Iris,	SAL : 20>,	
<DEPT : Computing,	NAME : Reuven,	SAL : 25,	SPS : Hanna>,
<DEPT : Computing,	NAME : Brian,	SAL : 30,	SPS : Annette>,
<DEPT : Maths,	NAME : Naomi,	SAL : 22,	SPS : Eli>,
<DEPT : Maths,	MGR : Naomi,	SEC : Eli>,	
<DEPT : Computing,	MGR : Brian,	SEC : Rachel>,	
<DEPT : Philosophy,	MGR : Dan,	SEC : Naomi>}	

Figure 2.1. The variable relation EMP.

DEPT	NAME	SAL	MGR	SEC	SPS
Computing	Iris	20			
Computing	Reuven	25			Hanna
Computing	Brian	30			Annette
Maths	Naomi	22			Eli
Maths			Naomi	Eli	
Computing			Brian	Rachel	
Philosophy			Dan	Naomi	

Figure 2.2. Tabular representation of EMP.

2.3. Encodings

We next formalise the notion of an ordering of a set $S \in \Psi$. Let $S = \{t_1, ..., t_n\} \in \Psi$, where $\forall$ $i \in I$, t_i is a record of the form $<A_1 : v_1, ..., A_m : v_m>$ or equivalently $\{(A_1, v_1), ..., (A_m, v_m)\}$.

We denote an ordering, say σ, of a set $S \in \Psi$ as the list $[t_1^\sigma, ..., t_n^\sigma]$ such that $S = \{t_1^\sigma, ..., t_n^\sigma\}$. Furthermore, we denote an ordering σ_i of a record t_i^σ as the list $[(A_1, v_1)^{\sigma_i}, ..., (A_m, v_m)^{\sigma_i}]$ such that $\forall$ $i \in I$, $t_i^\sigma = \{(A_1, v_1)^{\sigma_i}, ..., (A_m, v_m)^{\sigma_i}\}$. Now, if $\sigma = [t_1^\sigma, ..., t_n^\sigma]$ is an ordering of S and $\forall$ i $\in$ I, σ_i is an ordering of t_i^σ, then we call the list $[\sigma_1, ..., \sigma_n]$ *a full ordering of S* (or simply an ordering of S when no ambiguity arises) and denote it by S^σ.

Let Ω be the set of all full orderings of the sets $S \in \Psi$. Formally,

$$\Omega = \{S^\sigma \mid S^\sigma \text{ is a full ordering of } S \in \Psi\}.$$

Assume furthermore that $Delim \subset \Sigma$, where Delim = {[,], (,),,}, and let the concatenation of two strings $w, z \in \Sigma^*$ be written as wz.

Now let Π denote the set of all mappings from Ψ to Σ^*, where each mapping $\rho \in \Pi$ is of the form $\theta \circ \phi$ and such that it *satisfies* the following two conditions:

(1) ϕ is a total one-to-one mapping from Ψ to Ω such that $\forall$ $S \in \Psi$, $\phi(S) = S^\sigma$ (we call ϕ an *ordering function*); and

(2) θ is a one-to-one and onto mapping from $U \cup D$ to $(\Sigma - Delim)^*$ naturally extended to an isomorphism from Ω onto a restricted subset of Σ^* (which we denote by $\Sigma^*|_\Omega$) as follows:

$\forall$ $S^\sigma \in \Omega$, $\theta(S^\sigma) = [\theta(t_1^\sigma), ..., \theta(t_n^\sigma)]$ and
$\forall$ $i \in I$, $\theta(t_i^\sigma) = [(\theta(A_1), \theta(v_1))^{\sigma_i}, ..., (\theta(A_m), \theta(v_m))^{\sigma_i}]$.

From now on we call the mappings in Π *encodings*.

Now let Π^{-1} denote the set of mappings from Σ^* to Ψ, where each mapping $\rho^{-1} \in \Pi^{-1}$ is of the form $\gamma \circ \theta^{-1}$ such that it satisfies the following two conditions:

(1) $\exists$ $\rho = \theta \circ \phi \in \Pi$ such that θ^{-1} is the inverse mapping from $\Sigma^*|_\Omega$ to Ω of the isomorphism θ; and

(2) γ is the total and onto mapping from Ω to Ψ such that graph(γ) = $\cup_i$ {graph(ϕ_i^{-1}) | $\theta \circ \phi_i \in \Pi$} (we call γ the *forgetful function*).

Informally, γ *forgets* the ordering imposed on a set $S \in \Psi$ by an ordering function ϕ_i from Ψ to Ω. We note that γ is indeed a total and onto mapping, since its domain is Ω and $\forall\ \theta_1 \circ \phi_1, \theta_2 \circ \phi_2 \in \Pi$, $\forall\ S_1, S_2 \in \Psi$, if $S_1 \neq S_2$, then $\phi_1(S_1) \neq \phi_2(S_2)$. From now on we call the mappings in Π^{-1} *decodings*; the decoding $\rho^{-1} = \gamma \circ \theta^{-1}$ is called the inverse of an encoding $\rho = \theta \circ \phi \in \Pi$.

The following two technical lemmas will be used later on.

Lemma 2.1. Let $\rho_1, \rho_2 \in \Pi$. Then $\rho_2^{-1} \circ \rho_1 \circ \rho_1^{-1} = \rho_2^{-1}$.

Proof. Let $\rho_1 = \theta_1 \circ \phi_1$. Then $\rho_2^{-1} \circ \rho_1 \circ \rho_1^{-1} = \rho_2^{-1} \circ \theta_1 \circ \phi_1 \circ \gamma \circ \theta_1^{-1}$. Thus if $(\rho_1 \circ \rho_1^{-1})(w_1) = w_2$, i.e. w_2 is the result of reordering w_1, then $\rho_2^{-1}(w_1) = \rho_2^{-1}(w_2)$, since $\theta_1 \circ \theta_1^{-1}$ is the identity mapping on $\Sigma^*|_\Omega$. □

Lemma 2.2. Let $\rho_1, \rho_2 \in \Pi$, where $\rho_1 = \theta_1 \circ \phi_1$ and $\rho_2 = \theta_2 \circ \phi_2$. Then $\rho_1 \circ \rho_1^{-1} \circ \rho_2 = \theta_2 \circ \phi$, for some ordering function ϕ.

Proof. Let $\eta^\Psi = \rho_1^{-1} \circ \rho_2 = \gamma \circ \theta_1^{-1} \circ \theta_2 \circ \phi_2$ and let $\eta^\Omega = \theta_1^{-1} \circ \theta_2$ be isomorphisms from Ψ to Ψ and Ω to Ω, respectively. In addition, let $\phi = (\eta^\Omega)^{-1} \circ \phi_1 \circ \eta^\Psi$. It follows that $\rho_1 \circ \rho_1^{-1} \circ \rho_2 = \theta_1 \circ \eta^\Omega \circ \phi = \theta_2 \circ \phi$ as required, since $\eta^\Omega \circ (\eta^\Omega)^{-1}$ and $\theta_1 \circ \theta_1^{-1}$ are the identity mappings on Ω and $\Sigma^*|_\Omega$, respectively. □

The following assumption, which will be made from now on, asserts that one of the encodings in Π can be chosen as a standard encoding [GARE79].

The Standard Encoding Assumption. We assume that there exists an ordering function from Ψ to Ω, which we designate χ, that corresponds to a lexicographical order [HALM74] of $(U \cup D) \times (U \cup D)$. Furthermore, we assume that there exists a one-to-one and onto mapping from $U \cup D$ to $(\Sigma - \mathrm{Delim})^*$, which we designate ι, such that $\forall\ A \in U$, $\iota(A)$ = "A" and $\forall\ v \in D$, $\iota(v)$ = "v", where "A" and "v" are natural string representations, over $(\Sigma - \mathrm{Delim})^*$, of A and v, respectively (we omit the double quotes when no ambiguity arises).

As a consequence of this assumption $\iota \circ \chi$ is called a standard encoding. We note that according to the above assumption the semantics of ι are the same as those of the identity mapping on $U \cup D$. The following concept allows us to restrict the set of encodings only to encodings $\rho \in \Pi$ such that there is always an algorithm to convert $\iota \circ \chi$ to ρ and vice versa.

Mutually Convertible Encodings. An encoding $\rho \in \Pi$ is said to be *mutually convertible* to the encoding $\iota \circ \chi$ if both $\rho \circ (\iota \circ \chi)^{-1} \in TC$ and $(\iota \circ \chi) \circ \rho^{-1} \in TC$ hold. We let $\Pi^{TC} = \{\rho \mid \rho$ is mutually convertible to $\iota \circ \chi\}$.

The following proposition shows that the relation of being mutually convertible is an equivalence relation in Π^{TC} with a single equivalence class.

Proposition 2.3. $\forall\ \rho_1, \rho_2 \in \Pi^{TC}$, ρ_1 and ρ_2 are mutually convertible.

Proof. Let $\rho_1, \rho_2 \in \Pi^{TC}$. By the definition of mutually convertible encodings we have that $(\iota \circ \chi) \circ \rho_1^{-1} \in$ TC and $\rho_2 \circ (\iota \circ \chi)^{-1} \in$ TC. Thus $\rho_2 \circ (\iota \circ \chi)^{-1} \circ (\iota \circ \chi) \circ \rho_1^{-1} \in$ TC, since TC is closed under composition of mappings [ROGE67]. Hence $\rho_2 \circ \rho_1^{-1} \in$ TC, since $(\iota \circ \chi)^{-1} \circ (\iota \circ \chi)$ is the identity mapping on Ψ. □

From now on we use Π to denote Π^{TC} and Π^{-1} to denote $(\Pi^{TC})^{-1}$, since we are only interested in mutually convertible encodings. It now follows that all the encodings in Π are "reasonable" [GARE79], i.e. they are both "concise" and "decodable".

We now define a free encoding, which utilises the standard encoding assumption.

Definition 2.1. An encoding $\rho = \theta \circ \phi \in \Pi$ is called a *free encoding*, if $\theta = \iota$.

Hereafter, we let Π^{free} denote the set of all free encodings.

Example 2.2. In Figure 2.3 we show a free encoding of the variable relation EMP shown in Figure 2.1. On the other hand, in Figure 2.4 we show an encoding of EMP, which is not free.

[[(DEPT, Computing),	(NAME, Iris),	(SAL, 20)],	
[(DEPT, Computing),	(NAME, Brian),	(SAL, 30),	(SPS, Annette)],
[(DEPT, Computing),	(NAME, Reuven),	(SAL, 25),	(SPS, Hanna)],
[(DEPT, Maths),	(NAME, Naomi),	(SAL, 22),	(SPS, Eli)],
[(DEPT, Computing),	(MGR, Brian),	(SEC, Rachel)],	
[(DEPT, Maths),	(MGR, Naomi),	(SEC, Eli)],	
[(DEPT, Philosophy),	(MGR, Dan),	(SEC, Naomi)]]	

Figure 2.3. A free encoding of EMP.

[[(1, IRIS),	(2, COMPUTING),	(3, 20)],	
[(1, BRIAN),	(2, COMPUTING),	(3, 30),	(4, ANNETTE)],
[(1, NAOMI),	(2, MATHS),	(3, 22),	(4, ELI)],
[(1, REUVEN),	(2, COMPUTING),	(3, 25),	(4, HANNA)],
[(2, COMPUTING),	(5, BRIAN),	(6, RACHEL)],	
[(2, MATHS),	(5, NAOMI),	(6, ELI)],	
[(2, PHILOSOPHY),	(5, DAN),	(6, NAOMI)]]	

Figure 2.4. An encoding of EMP.

The next definition sets up some useful notation.

Definition 2.2. An *encoding oracle* is a mapping ENC from $\Pi \times \Psi$ onto $\Sigma^*|_\Omega$. Thus an encoding oracle takes as input an encoding $\rho \in \Pi$ and a set $S \in \Psi$ and returns a string in $\Sigma^*|_\Omega$. A *decoding oracle* is a mapping DEC from $\Pi^{-1} \times \Sigma^*|_\Omega$ onto Ψ. Thus a decoding oracle takes as input a decoding $\rho^{-1} \in \Pi^{-1}$ and a string in $\Sigma^*|_\Omega$ and returns a set $S \in \Psi$. An ordered pair (ENC, DEC) is called an *encoding-decoding oracle* (or simply an ed-oracle).

2.4. Computable Database Queries

We begin this subsection with an example.

Example 2.3. We now give several examples of queries, which intuitively are computable, and whose input is the variable relation EMP of Example 2.1.

(1) Project EMP onto the set of attributes {DEPT, NAME}.

(2) Select from EMP the records whose DEPT value is Computing.

(3) Return the nth record in EMP (with respect to an encoding which is used to encode EMP) if $|EMP| \geq n$, otherwise return {<>}.

(4) Select from EMP the records, t, where $|t| = n$.

(5) Return {<>} if $|EMP| \geq n$, otherwise return $\emptyset$.

(6) Select from EMP all records t_i such that there exists another record t_j in EMP such that $t_i \neq t_j$ and $t_i \cap t_j \neq \emptyset$.

Informally, a computable query is a mapping, $\tau \in \Phi$, that can be computed via a Turing-computable mapping δ and an ed-oracle, which encodes the input of τ into a string using an encoding $\rho \in \Pi$ and decodes the output of δ using the decoding ρ^{-1}. Figure 2.5 shows the commutative diagram describing the semantics of a computable query.

.

$$\begin{tikzcd}
\Psi \arrow[r, "\tau"] \arrow[d, "\rho"'] & \Psi \\
\Sigma^*|\Omega \arrow[r, "\delta"'] & \Sigma^*|\Omega \arrow[u, "\rho^{-1}"']
\end{tikzcd}$$

Figure 2.5. Commutative diagram.

Definition 2.3. Let (ENC, DEC) be an ed-oracle. A mapping $\tau \in \Phi$ is a *computable query* (or simply a query) if

$$\exists\, \delta \in TC, \exists\, \rho \in \Pi, \forall\, S \in \Psi, \tau(S) = DEC(\rho^{-1}, \delta(ENC(\rho, S))).$$

In this definition we consider $\tau(S) = DEC(\rho^{-1}, \delta(ENC(\rho, S)))$ to obtain if either both $\tau(S)$ and $DEC(\rho^{-1}, \delta(ENC(\rho, S)))$ are defined and equal or both $\tau(S)$ and $DEC(\rho^{-1}, \delta(ENC(\rho, S)))$ are undefined.

As an abbreviation to the above equation we say that τ is *realised* via δ and ρ; at times we simply say that τ is realised via δ meaning that $\exists\ \rho \in \Pi$, such that τ is realised via δ and ρ. In the sequel we let CQ denote the set of all computable queries.

Example 2.4. We observe that it can be verified that all the queries given in Example 2.3 are in fact computable queries according to Definition 2.3.

The following proposition shows that if a computable query τ is realised via δ and ρ then for any other encoding $\rho_i \in \Pi$ we can effectively find a Turing-computable mapping δ_i such that τ is realised via δ_i and ρ_i.

Proposition 2.4. Let $\tau \in$ CQ. Then $\forall\ \rho_i \in \Pi$, $\exists\ \delta_i \in$ TC such that τ is realised via δ_i and ρ_i.

Proof. Assume that τ is realised via δ and ρ. Let $\delta_i = \rho_i \circ \rho^{-1} \circ \delta \circ \rho \circ \rho_i^{-1}$. By Proposition 2.3 and the fact that TC is closed under composition of mappings it follows that $\delta_i \in$ TC. Therefore τ is realised via δ_i and ρ_i as required. □

We observe that Definition 2.3 of a computable query is more general than the analogous definition in [CHAN80]. The reason for this is that we encode a set as a list of strings prior to the evaluation of a computable query and thus our computable queries include all the natural queries resulting from viewing a set as a list of strings, i.e. when imposing an ordering on the set. In the next section we characterise the notion of a computable query as defined in [CHAN80] in terms of a subclass of CQ.

3. Encoding-Independent Computable Queries

In this section we investigate the relationship between Definition 2.3 of a computable query and the definition of a *generic computable query* (or simply a generic query) given in [CHAN80]. (We note that in [CHAN80] genericity is called the consistency criterion.) Informally, a computable query is generic if whenever two input sets to the query are isomorphic then the two outputs of the query are also isomorphic. In Example 2.3 queries (4), (5) and (6) are generic, while query (3) is not; queries (1) and (2) are also not generic but they are in fact C-generic (see the note immediately after the ensuing definition). The formal definition of a generic computable query, recast in our context and notation, follows.

Definition 3.1. A mapping $\tau \in \Phi$ is *a generic computable query* (or simply a generic query) if

(1) τ is an *order-independent* computable query (see Definition 3.3), and

(2) τ is *generic*, i.e. for all isomorphisms $\eta \in \Phi$, $\forall\ S \in \Psi$, $\eta(\tau(S)) = \tau(\eta(S))$.

We note that condition (1) of Definition 3.1 is stated in [CHAN80] as: τ is a partially recursive function. We call our interpretation of this statement, as given in (1) of Definition 3.1, the *order-independence assumption*. We further note that in [ABIT90] a more general definition of genericity, called *C-genericity*, is considered. In our context C-genericity means that a finite set, $C \subset U \cup D$, of distinguished attributes and values, which are the constants explicitly named in the query, are mapped by the isomorphism η to themselves (i.e. the isomorphism is the identity mapping on C). This takes care of queries such as "select the records in EMP with NAME = Naomi" and "insert the record <NAME : David, DEPT : Philosophy> into EMP ". We observe that in our case a query such as "project the records in EMP onto the attributes NAME and SAL", which expresses a projection, is C-generic, where C = {NAME, SAL} is the set of attributes explicitly named in the query. In this section, for simplicity, we have assumed that $C = \emptyset$; the more general case, where $C \neq \emptyset$, is investigated in Section 4.

There has been much interest in generic computable queries since the seminal work of [AHO79] and [CHAN80]; a summary of the research in this area was presented in [CHAN88]. In order to motivate generic queries we give four quotes, which stipulate the principles that queries ought to satisfy. The first two are from [CHAN80, p.158] and the last two are from [AHO79, p.111]:

(1.1) "the result of a query should be independent of the representation of the data in a data base"; and

(1.2) (the result of a query) "should treat the elements of the database as uninterpreted objects."

(2.1) "one wants the query language to be sufficiently high level that it deals with relations as sets of tuples, meaning that the order in which the tuples are considered should not influence the result"; and

(2.2) "queries should preserve symmetries that exist amongst the values that appear in argument relations."

The definition of a generic query is intended to capture the above principles. However, as far as the authors are aware, these principles have not been lucidly explained in terms of generic queries although the fact that they are satisfied is "well known".

In the rest of this section we present an explicit account of the above principles in terms of a subclass of the set of computable queries, CQ, and then prove the equivalence of this subclass with the class of generic queries.

Informally, a computable query is encoding-independent if its computation does not depend on any particular encoding $\rho \in \Pi$ used by an ed-oracle. That is, intuitively a computable query is encoding-independent if the diagram shown in Figure 2.5 commutes for all encodings $\rho \in \Pi$.

Definition 3.2. A computable query $\tau \in$ CQ is *encoding-independent* if

$$\exists\, \delta \in TC, \forall\, \rho \in \Pi, \forall\, S \in \Psi, \tau(S) = DEC(\rho^{-1}, \delta(ENC(\rho, S))).$$

Hereafter, we let EI denote the set of all encoding-independent computable queries.

An encoding-independent query (which is invariant with respect to the encodings $\rho = \theta \circ \phi \in \Pi$) can be seen to be:

(1) invariant with respect to the ordering functions ϕ which impose an ordering on the records and ordered pairs thereof in the input set, say S, that is, variable relations are treated as sets of records (1.1 and 2.1);

(2) invariant with respect to isomorphisms θ, which impose an internal representation on the attributes and data values in the records of the input set, i.e. on adom(S) (1.1 and 1.2); and

(3) preserving symmetries that exist amongst the attributes and values in adom(S), since θ is an isomorphism (2.2).

We next give two auxiliary definitions, namely Definitions 3.3 and 3.4, which allow us to view encoding-independence as the combination of two orthogonal concepts, i.e. order-independence and isomorphism-independence.

Informally, a computable query is order-independent if its computation does not depend on any ordering function ϕ of an encoding $\rho = \theta \circ \phi \in \Pi$ used by an ed-oracle. That is, intuitively a computable query that is realised via δ and $\rho = \theta \circ \phi$ is order-independent if the diagram shown in Figure 2.5 commutes for all encodings $\rho_i = \theta_i \circ \phi_i \in \Pi$ with $\theta = \theta_i$.

Definition 3.3. We define a Turing-computable mapping $\delta \in$ TC to be *order-independent* if

$$\forall \rho_i = \theta \circ \phi_i \in \Pi, \forall \rho = \theta \circ \phi \in \Pi, \forall S \in \Psi,$$
$$\mathrm{DEC}(\rho_i^{-1}, \delta(\mathrm{ENC}(\rho_i, S))) = \mathrm{DEC}(\rho^{-1}, \delta(\mathrm{ENC}(\rho, S))).$$

In this definition we consider $\mathrm{DEC}(\rho_i^{-1}, \delta(\mathrm{ENC}(\rho_i, S))) = \mathrm{DEC}(\rho^{-1}, \delta(\mathrm{ENC}(\rho, S)))$ to obtain if either both sides of this equation are defined and equal or if both sides of the equation are undefined. We note that in the above equation $\rho_i^{-1} = \rho^{-1}$.

Hereafter, we let OITC denote the set of all order-independent Turing-computable mappings. A computable query $\tau \in$ CQ is *order-independent* if $\exists\, \delta \in$ OITC such that τ is realised via δ.

(Recall the order-independence assumption made after Definition 3.1, which stated that a generic computable query is order-independent.) We note that order-independence is a property of the Turing-computable mapping δ that realises τ. Hereafter, we let OI denote the set of all order-independent computable queries. We observe that in Example 2.3 only query (3) is not order-independent and thus OI $\subset$ CQ.

Informally, a computable query is isomorphism-independent if its computation does not depend on any isomorphism θ of an encoding $\rho = \theta \circ \phi \in \Pi$ used by an ed-oracle. That is, intuitively a computable query that is realised via δ and $\rho = \theta \circ \phi$ is isomorphism-independent if the diagram shown in Figure 2.5 commutes for all encodings $\rho_i = \theta_i \circ \phi_i \in \Pi$ with $\phi = \phi_i$.

Definition 3.4. We define a Turing-computable mapping $\delta \in$ TC to be *isomorphism-independent* if

$$\forall \rho_i = \theta_i \circ \phi \in \Pi, \forall \rho = \theta \circ \phi \in \Pi, \forall S \in \Psi,$$
$$\mathrm{DEC}(\rho_i^{-1}, \delta(\mathrm{ENC}(\rho_i, S))) = \mathrm{DEC}(\rho^{-1}, \delta(\mathrm{ENC}(\rho, S))).$$

Again in this definition we consider $DEC(\rho_i^{-1}, \delta(ENC(\rho_i, S))) = DEC(\rho^{-1}, \delta(ENC(\rho, S)))$ to obtain if either both sides of this equation are defined and equal or if both sides of the equation are undefined.

Hereafter, we let IITC denote the set of all isomorphism-independent Turing-computable mappings. A computable query $\tau \in$ CQ is *isomorphism-independent* if $\exists\ \delta \in$ IITC such that τ is realised via δ.

We note that isomorphism-independence is a property of the Turing-computable mapping δ that realises τ. From Definitions 3.2, 3.3 and 3.4 it follows that $\tau \in$ EI if and only if $\exists\ \delta \in$ TC such that τ is realised via δ and $\delta \in$ OITC $\cap$ IITC. Hereafter, we let II denote the set of all isomorphism-independent computable queries. We observe that in Example 2.3 only queries (1) and (2) are not isomorphism-independent and thus II $\subset$ CQ. We also observe that EI $\subset$ II and EI $\subset$ OI both hold, since in Example 2.3 query (3) is isomorphism-independent but not encoding-independent and queries (1) and (2) are order-independent but not encoding-independent. The stronger result that EI = OI $\cap$ II is shown at the end of this section.

The following theorem shows that the concepts of genericity and encoding-independence with reference to a computable query are equivalent, thus providing us with an insight into the standard definition of a generic computable query originally given in [CHAN80].

Theorem 3.1. A computable query $\tau \in$ CQ is encoding-independent if and only if it is a generic computable query.

Proof. Let $\eta \in \Phi$ be an isomorphism, $S \in \Psi$, and assume that τ is realised via $\delta \in$ OITC and an encoding $\rho \in \Pi$.

(*If*): Without loss of generality let $\eta = \rho_1^{-1} \circ \rho_2$, where $\rho_1 = \theta_1 \circ \phi_1$, $\rho_2 = \theta_2 \circ \phi_2 \in \Pi$. Also, let $\tau(S) = DEC(\rho_1^{-1}, \delta(ENC(\rho_1, S)))$, i.e. we assume that $\rho = \rho_1$. Furthermore, $\tau(\eta(S)) = DEC(\rho_1^{-1}, \delta(ENC(\rho_1, \eta(S)))) = DEC(\rho_1^{-1}, \delta(ENC(\rho_1 \circ \eta, S)))$. Thus we obtain $\tau(\eta(S)) = DEC(\rho_1^{-1}, \delta(ENC(\theta_2 \circ \phi, S)))$ for some ordering function ϕ, on using Lemma 2.2.

On using genericity we have that $\tau(\eta(S)) = \eta(\tau(S))$ and thus $\eta^{-1}(\tau(\eta(S))) = \tau(S)$. Thus, on using Lemma 2.1, $\tau(S) = \eta^{-1}(DEC(\rho_1^{-1}, \delta(ENC(\theta_2 \circ \phi, S)))) = DEC(\rho_2^{-1} \circ \rho_1 \circ \rho_1^{-1}, \delta(ENC(\theta_2 \circ \phi, S))) = DEC(\rho_2^{-1}, \delta(ENC(\theta_2 \circ \phi, S)))$. Finally, by the order-independence assumption, $\tau(S) = DEC(\rho_2^{-1}, \delta(ENC(\theta_2 \circ \phi_2, S))) = DEC(\rho_2^{-1}, \delta(ENC(\rho_2, S)))$ as required.

(*Only if*): By Definitions 3.2 and 3.3 it follows that $\tau \in$ OI. It therefore remains to show that τ is generic.

By Definition 3.2 of encoding-independence we have that $\tau(S) = DEC(\rho^{-1}, \delta(ENC(\rho, S))) = DEC(\eta^{-1} \circ \rho^{-1}, \delta(ENC(\rho \circ \eta, S)))$, since $\rho \circ \eta$ is also an encoding in Π. Moreover, $\eta(\tau(S)) = \eta(DEC(\eta^{-1} \circ \rho^{-1}, \delta(ENC(\rho \circ \eta, S)))) = DEC(\eta \circ \eta^{-1} \circ \rho^{-1}, \delta(ENC(\rho \circ \eta, S))) = DEC(\rho^{-1}, \delta(ENC(\rho \circ \eta, S))) = DEC(\rho^{-1}, \delta(ENC(\rho, \eta(S)))) = \tau(\eta(S))$ as required. $\square$

The following corollary shows some important consequences related to Theorem 3.1. In part (1) we show that $\delta \in$ TC of Definition 3.2 can be chosen to be any $\delta \in$ OITC. On the other hand, in part (2) we show the interesting result that δ cannot be chosen to be any $\delta \in$ IITC.

Corollary 3.2.

(1) If $\tau \in$ EI then $\forall\ \delta \in$ OITC such that τ is realised via δ, it is true that

$$\forall\ \rho \in \Pi, \forall\ S \in \Psi, \tau(S) = DEC(\rho^{-1}, \delta(ENC(\rho, S))).$$

(2) $\exists\ \tau \in$ EI, $\exists\ \delta \in$ IITC, such that τ is realised via δ but $\delta \notin$ OITC.

(3) EI = OI $\cap$ II.

Proof. (1) Assume that $\tau \in$ EI, then by the only if part of Theorem 3.1 τ is a generic computable query and thus $\tau \in$ OI. The result now follows, since the if part of the proof of Theorem 3.1 is not dependent on any particular choice of $\delta \in$ TC, in this case $\delta \in$ OITC.

(2) Let $\tau \in$ CQ be the query that returns {<>} whenever its input, $S \in \Psi$, is such that $|S| = 3$ and $\exists\ t_i, t_j \in S, t_i \neq t_j$ and $t_i \cap t_j \neq \emptyset$, otherwise it returns $\emptyset$. It can easily be verified that $\tau \in$ EI.

We now exhibit a $\delta \in$ IITC and two ordering functions ϕ_1 and ϕ_2 such that τ is realised via δ and $\rho_1 = \theta \text{ o } \phi_1$, but τ is not realised via δ and $\rho_2 = \theta \text{ o } \phi_2$, thus implying that $\delta \notin$ OITC.

Let δ be the Turing-computable mapping that outputs [[]] (i.e. it outputs the string representing the singleton containing the empty record) if the number of records in S, the input to τ, is three and the intersection of the first and third records in S is not empty, otherwise it outputs [] (i.e. it outputs the string representing the empty set of records).

Now, let ϕ_1 be an ordering function such that whenever $S \in \Psi$, $|S| = 3$ and $\exists\ t_i, t_j \in S, t_i \neq t_j$ and $t_i \cap t_j \neq \emptyset$, then $\phi_1(S) = [[t_1, t_2, t_3]]$ satisfies the constraint: $t_1 \cap t_3 \neq \emptyset$. Next, let ϕ_2 be any other ordering function that does not satisfy the above constraint whenever $\exists\ t_i, t_j \in S, t_i \cap t_j = \emptyset$. The result then follows.

(3) EI $\subseteq$ OI $\cap$ II follows from Definition 3.2 and the definitions of OI and II. It remains to show that OI $\cap$ II $\subseteq$ EI. Let $\tau \in$ OI $\cap$ II and assume that τ is realised via $\delta_1 \in$ IITC and $\rho_1 = \theta_1 \text{ o } \phi_1$ and also that τ is realised via $\delta_2 \in$ OITC and $\rho_2 = \theta_2 \text{ o } \phi_2$. In order to show that $\tau \in$ EI it is sufficient to prove that τ is a generic query, which concludes the result by the if part of Theorem 3.1.

Let η be a one-to-one and onto mapping from $U \cup D$ to $U \cup D$, and let η^Ψ and η^Ω be the natural extensions of η to isomorphisms from Ψ to Ψ and from Ω to Ω, respectively. Also, let $S \in \Psi$. Now, we have that $\tau(S) = DEC(\rho_1^{-1}, \delta_1(ENC(\rho_1, S))) = DEC(\gamma \text{ o } \theta_1^{-1}, \delta_1(ENC(\theta_1 \text{ o } \phi_1, S))) = DEC(\gamma \text{ o } (\eta^\Omega)^{-1} \text{ o } \theta_2^{-1}, \delta_1(ENC(\theta_2 \text{ o } \eta^\Omega \text{ o } \phi_1, S)))$, since τ is realised via δ_1 and $\theta_1 \text{ o } \phi_1$ with $\delta_1 \in$ IITC. (Recall that γ is the forgetful function.)

We also have that $\eta^\Psi(\tau(S)) = \eta^\Psi(DEC(\gamma \text{ o } (\eta^\Omega)^{-1} \text{ o } \theta_2^{-1}, \delta_1(ENC(\theta_2 \text{ o } \eta^\Omega \text{ o } \phi_1, S)))) = DEC(\eta^\Psi \text{ o } \gamma \text{ o } (\eta^\Omega)^{-1} \text{ o } \theta_2^{-1}, \delta_1(ENC(\theta_2 \text{ o } \eta^\Omega \text{ o } \phi_1, S))) = DEC(\gamma \text{ o } \theta_2^{-1}, \delta_1(ENC(\theta_2 \text{ o } \eta^\Omega \text{ o } \phi_1, S)))$, on using an argument similar to that used in Lemma 2.1.

Furthermore, $DEC(\gamma \text{ o } \theta_2^{-1}, \delta_1(ENC(\theta_2 \text{ o } \eta^\Omega \text{ o } \phi_1, S))) = DEC(\rho_2^{-1}, \delta_1(ENC(\theta_2 \text{ o } \eta^\Omega \text{ o } \phi_1, S))) = DEC(\rho_2^{-1}, \delta_1(ENC(\theta_2 \text{ o } \phi, \eta^\Psi(S))))$, for some ordering function ϕ, on using an argument similar to that used in Lemma 2.2. It now follows that $DEC(\rho_2^{-1}, \delta_1(ENC(\theta_2 \text{ o } \phi, \eta^\Psi(S)))) = DEC(\rho_2^{-1}, \delta_2(ENC(\theta_2 \text{ o } \phi, \eta^\Psi(S))))$, since τ is realised via δ_2 and $\theta_2 \text{ o } \phi_2$ with $\delta_2 \in$ OITC.

Finally, it follows that $\mathrm{DEC}(\rho_2^{-1}, \delta_2(\mathrm{ENC}(\theta_2 \circ \phi, \eta^{\Psi}(S)))) = \mathrm{DEC}(\rho_2^{-1}, \delta_2(\mathrm{ENC}(\theta_2 \circ \phi_2, \eta^{\Psi}(S)))) = \mathrm{DEC}(\rho_2^{-1}, \delta_2(\mathrm{ENC}(\rho_2, \eta^{\Psi}(S)))) = \tau(\eta^{\Psi}(S))$, since $\delta_2 \in$ OITC. □

4. Free-Encoding Computable Queries

Herein we define a natural class of computable queries, namely the class of free-encoding computable queries, which we denote by FE. For computable queries in FE an ed-oracle uses only free encodings. (Recall Definition 2.1 of a free encoding.) This class of computable queries corresponds to a natural class of computable queries, since attributes and values are naturally represented as strings in the real world. We show that FE = CQ and thus there is no loss of computational expressiveness when restricting ourselves to free encodings. We then define a subclass of FE, the class of encoding-independent free-encoding computable queries, which we denote by EIFE, and show that EIFE = OI, so OI does not lose any of its computational expressiveness when it is restricted to free encodings. We then show that the class which contains the set of all C-generic computable queries, for all finite sets C [ABIT90], and EIFE are equivalent. This result is obtained by exhibiting a complete lattice, partially ordered by set inclusion. This complete lattice comprises sets of computable queries together with the class EIFE of computable queries; each of these sets is the set of C-generic computable queries, for some finite set C. The class EIFE of computable queries is shown to be the top element of this lattice and the class EI of computable queries is shown to be the bottom element thereof. The significance of this result lies in the fact that we conclude that in order to show that a database language expresses the class of all C-generic computable queries, such as the database language detDL [ABIT90], we need only consider computable queries in EIFE. We thus claim that the class EIFE of computable queries is the most natural class of computable queries that a database language ought to express in order to be considered computationally complete.

Definition 4.1. A computable query $\tau \in$ CQ is *free-encoding* if

$$\exists\, \delta \in \mathrm{TC}, \exists\, \rho \in \Pi^{\mathrm{free}}, \forall\, S \in \Psi, \tau(S) = \mathrm{DEC}(\rho^{-1}, \delta(\mathrm{ENC}(\rho, S))).$$

Hereafter, we let FE denote the set of all free-encoding computable queries. An immediate consequence of Definition 4.1 and Proposition 2.4 is that FE = CQ, i.e. we can restrict ourselves to free encodings without any loss of computational expressiveness. Hence FE is a natural class of computable queries to consider, since it corresponds to the situation of using *cursors* [ULLM88], when a query language such as SQL is being embedded in a general purpose programming language such as COBOL or C. This situation highlights the disadvantages of embedded query languages, since both order-independence and isomorphism-independence are sacrificed.

Informally, a free-encoding computable query is encoding-independent if its computation does not depend on any particular free encoding $\rho \in \Pi^{\mathrm{free}}$ used by an ed-oracle.

Definition 4.2. A free-encoding computable query $\tau \in$ CQ is *encoding-independent* if

$\exists\, \delta \in$ OITC, $\forall\, \rho \in \Pi^{\text{free}}$, $\forall\, S \in \Psi$, $\tau(S) = \text{DEC}(\rho^{-1}, \delta(\text{ENC}(\rho, S)))$.

Hereafter, we call an encoding-independent free-encoding computable query simply a *free-encoding-independent* computable query and we let EIFE denote the set of all such computable queries. An immediate consequence of Definition 4.2 is that EI $\subset$ EIFE, since $\Pi^{\text{free}} \subset \Pi$; in Example 2.3 queries (1) and (2) are in EIFE but not in EI. Another immediate consequence of Definition 4.2 is that EIFE = OI, since EIFE $\subseteq$ OI follows immediately from Definition 4.2 and the definition of OI, and OI $\subseteq$ EIFE follows from Proposition 2.4, since if $\tau \in$ OI is realised via $\delta \in$ OITC and ρ, then $\tau \in$ EIFE and it is realised via $\rho_i \circ \rho^{-1} \circ \delta \circ \rho \circ \rho_i^{-1}$, with $\rho_i \in \Pi^{\text{free}}$. This fact implies that the class OI of computable queries is not dependent on the isomorphism θ of Definition 3.3 as long as θ is fixed. Hence OI does not lose any of its computational expressiveness when it is restricted to free encodings.

We recall that in the discussion after Definition 3.1 we have informally defined C-generic computable queries. We next define C-encodings and C-encoding-independent queries, and then show that C-generic computable queries are equivalent to C-encoding-independent queries. As before we let $C \in P(U \cup D)$ be a finite set of attributes and values.

Definition 4.3. An encoding $\rho = \theta \circ \phi \in \Pi$ is called a *C-encoding* if $\forall\, d \in C$, $\theta(d) = \iota(d)$.

In the sequel we let Π^{C} denote the set all C-encodings. The following proposition follows from Definition 4.3.

Proposition 4.1. Let $C_1, C_2 \in P(U \cup D)$. Then $C_1 \subseteq C_2$ if and only if $\Pi^{C_2} \subseteq \Pi^{C_1}$.

Informally, a C-encoding computable query is C-encoding-independent if its computation does not depend on any particular C-encoding $\rho \in \Pi^{C}$ used by an ed-oracle.

Definition 4.4. A computable query $\tau \in$ CQ is *C-encoding-independent* if

$\exists\, \delta \in$ OITC, $\forall\, \rho \in \Pi^{C}$, $\forall\, S \in \Psi$, $\tau(S) = \text{DEC}(\rho^{-1}, \delta(\text{ENC}(\rho, S)))$.

Hereafter, we let C-EI denote the set of all C-encoding-independent computable queries. We observe that in Example 2.3 query (1) is {DEPT, NAME}-encoding-independent and query (2) is {DEPT, Computing}-encoding-independent. An immediate consequence of Definition 4.4 is that C-EI $\subset$ (EIFE = OI), since $\Pi^{\text{free}} \subset \Pi^{C}$.

The following proposition follows from Theorem 3.1 and Definition 4.4.

Proposition 4.2. A computable query $\tau \in$ CQ is C-encoding-independent if and only if it is C-generic. $\square$

The following theorem establishes the connection between C-encodings and C-encoding-independent queries.

Theorem 4.3. Let $C_1, C_2 \in P(U \cup D)$. Then $\Pi^{C_2} \subseteq \Pi^{C_1}$ if and only if C_1-EI $\subseteq C_2$-EI.

Proof. (*If*): If $\Pi^{C_2} \not\subseteq \Pi^{C_1}$, then by Proposition 4.1. $C_1 \not\subseteq C_2$. It follows that $\exists\, c_1 \in C_1$ such that $c_1 \notin C_2$. In order to prove the result we need to show that C_1-EI $\not\subseteq C_2$-EI. We show this by constructing a C_1-encoding-independent query $\tau_1 \in C_1$-EI such that $\tau_1 \notin C_2$-EI. Assume that τ_1 is realised via $\delta_1 \in$ OITC and $\rho_1 \in \Pi^{C_1}$. We need to show that $\nexists\, \delta_2 \in$ OITC such that τ_1 is realised via δ_2 and $\rho_2 \in (\Pi^{C_2} - \Pi^{C_1})$. Without loss of generality let $\rho_1 = \theta_1 \circ \phi_1$ with $\theta_1(c_1) = \iota(c_1)$ and let $\rho_2 = \theta_2 \circ \phi_2$ with $\theta_2(c_1) \neq \iota(c_1)$. It is sufficient to show that $\exists\, \rho_3 \in \Pi^{C_2}$, $\exists\, S \in \Psi$ such that $DEC(\rho_1^{-1}, \delta_1(ENC(\rho_1, S))) \neq DEC(\rho_3^{-1}, \delta_2(ENC(\rho_3, S)))$. Let τ_1 be such that it explicitly names c_1, i.e. if $c_1 \in U$, then τ_1 could, for example, project records onto c_1 and if $c_1 \in D$, then τ_1 could, for example, select records with an attribute value equal to c_1. It follows that δ_1 must explicitly name $\iota(c_1)$ in order to realise τ_1. Now, since $\theta_2(c_1) \neq \iota(c_1)$, it must be the case that δ_2 explicitly names $\theta_2(c_1)$ if indeed τ_1 is realised via δ_2 and ρ_2. Let $\rho_3 = \theta_3 \circ \phi_3$ be such that $\theta_3(c_1) \neq \iota(c_1)$ and $\theta_3(c_1) \neq \theta_2(c_1)$. We can now assume without loss of generality that δ_2 does not explicitly name $\theta_3(c_1)$, since δ_2 can explicitly name only a finite number of strings. The result now follows, since τ_1 is not realised via δ_2 and ρ_3 and thus $\tau_1 \notin C_2$-EI as required. □

(*Only if*): The result follows from Definition 4.4. □

The following corollary is an immediate consequence of the above theorem and Proposition 4.1.

Corollary 4.4. Let $C_1, C_2 \in P(U \cup D)$. Then C_1-EI = C_2-EI if and only if $C_1 = C_2$. □

In the sequel we let $\langle \mathbf{P}(\Pi), \subseteq \rangle$ be a partially ordered set, where **P** denotes the powerset operator, and let $\Pi^{\forall C} = \{\Pi^C \mid C \in P(U \cup D)\}$, i.e. $\Pi^{\forall C}$ is the set of all C-encodings for all finite sets C. We observe that $\Pi^{\forall C} \subset \mathbf{P}(\Pi)$.

The next lemma shows that Π^{free} is the *greatest lower bound* (glb) and that Π is the *least upper bound* (lub) of $\Pi^{\forall C}$.

Lemma 4.5. Π^{free} and Π are, respectively, the glb and the lub of the set $\Pi^{\forall C}$ with respect to the partially ordered set $\langle \mathbf{P}(\Pi), \subseteq \rangle$.

Proof. We first show that Π^{free} is the glb of $\Pi^{\forall C}$. We observe that $\forall\, C \in P(U \cup D)$, $\Pi^{free} \subseteq \Pi^C$. It remains to show that if $\Pi' \subseteq \Pi$ and $\forall\, C \in P(U \cup D)$, $\Pi' \subseteq \Pi^C$, then $\Pi' \subseteq \Pi^{free}$. Suppose this is not the case. Let $\rho' = \theta' \circ \phi' \in \Pi'$. Now, since by assumption $\theta' \neq \iota$, $\exists\, d \in U \cup D$ such that $\theta'(d) \neq \iota(d)$. It follows, by Definition 4.3, that $\forall\, C \in P(U \cup D)$ such that $d \in C$, ρ' is not a C-encoding. This leads to a contradiction, i.e. $\exists\, C \in P(U \cup D)$ such that $\Pi' \not\subseteq \Pi^C$.

We next show that Π is the lub of $\Pi^{\forall C}$. We observe that $\forall\, C \in P(U \cup D)$, $\Pi^C \subseteq \Pi$ holds. It remains to show that if $\Pi' \subseteq \Pi$ and $\forall\, C \in P(U \cup D)$, $\Pi^C \subseteq \Pi'$, then $\Pi \subseteq \Pi'$, i.e. $\Pi' = \Pi$. Since $\Pi' = \Pi^{\emptyset} = \Pi$, the result follows on using Proposition 4.1. □

The following theorem establishes a *complete lattice* from $\{\Pi^{free}\}$ and $\Pi^{\forall C}$.

Theorem 4.6. $\{\Pi^{free}\} \cup \Pi^{\forall C}$ is a complete lattice with respect to the partially ordered set $\langle \mathbf{P}(\Pi), \subseteq \rangle$.

Proof. Let K = {1,...,p} be an index set, where either $p \in \omega$ or $K = \omega$. We first observe that $\Pi \in \{\Pi^{free}\} \cup \Pi^{\forall C}$, since $\Pi = \Pi^{\varnothing}$. Furthermore, by Lemma 4.5, Π is the lub of $\{\Pi^{free}\} \cup \Pi^{\forall C}$ with respect to $\langle \mathbf{P}(\Pi), \subseteq \rangle$. By Corollary 2.17 [DAVE90] it remains to show that for every non-empty family $\{\Pi_k\}$ in $\{\Pi^{free}\} \cup \Pi^{\forall C}$, it is true that $\cap_{k \in K} \Pi_k \in \{\Pi^{free}\} \cup \Pi^{\forall C}$.

There are two cases to consider. Firstly, if $\exists\, k \in K$ such that $\Pi_k = \Pi^{free}$, then the result follows by Lemma 4.5, since $\cap_{k \in K} \Pi_k = \Pi^{free}$. Secondly, if $\nexists\, k \in K$ such that $\Pi_k = \Pi^{free}$, then $\cap_{k \in K} \Pi_k = \Pi^{C'}$, where $C' = \cup_{k \in K} \{C_k \mid \Pi_k = \Pi^{C_k}\}$, on using Proposition 4.1. □

Let $EI^{\forall C} = \{C\text{-}EI \mid C \in P(U \cup D)\}$. The next corollary follows from the duality principle [DAVE90] for partially ordered sets and Theorems 4.3 and 4.6.

Corollary 4.7. $\{EIFE\} \cup EI^{\forall C}$ is a complete lattice with respect to the partially ordered set $\langle \mathbf{P}(EIFE), \subseteq \rangle$ with EI being its glb and EIFE being its lub. □

The next corollary shows that the class EIFE of computable queries captures exactly the class of all C-generic computable queries, for all finite sets $C \in P(U \cup D)$, i.e. it captures the class $\cup \{C\text{-}EI \mid C \in P(U \cup D)\}$.

Corollary 4.8. Let $\tau \in CQ$. Then $\tau \in EIFE$ if and only if $\exists\, C \in P(U \cup D)$ such that $\tau \in C\text{-}EI$.

Proof. (*If*): It follows from Corollary 4.7.

(*Only if*): By the duality principle for partially ordered sets, $\cup$ is the lub operator for the complete lattice $\{EIFE\} \cup EI^{\forall C}$. Thus $\cup \{C\text{-}EI \mid C \in P(U \cup D)\} \in \{EIFE\} \cup EI^{\forall C}$. On using Lemma 4.5 and the duality principle for ordered sets, we conclude that EIFE is the lub of $EI^{\forall C}$ and thus $EIFE = \cup \{C\text{-}EI \mid C \in P(U \cup D)\}$ as required. □

Suppose that τ is realised via δ and ρ. It is interesting to note that the only if part of Corollary 4.8 can also be argued by the fact that δ can explicitly name only a finite number of strings, since it is Turing-computable, and thus we must be able to describe it in finite terms. The above proof is at a high level, since it does not delve into the mechanics of realising τ.

Corollary 4.8 entails a very important consequence which we now discuss. Both [ABIT90] and [HULL93] consider the class of all C-generic computable queries, i.e. by Proposition 4.2 the class $\cup \{C\text{-}EI \mid C \in P(U \cup D)\}$. Call a database language *computationally complete* if it expresses exactly the class of all C-generic computable queries. Now, by Corollary 4.8, a database language is computationally complete if and only if it expresses exactly the class EIFE of computable queries. Thus we claim that EIFE is the most natural class of computable queries that a database language ought to express in order to be considered computationally complete. For example, the database language detDL [ABIT90] is computationally complete and consequently expresses exactly the class EIFE of computable queries. In addition, in order to prove the computational completeness of any specific database language only free encodings need be considered. This can result in simpler completeness proofs.

5. An Equivalence Relation on Computable Queries

The aim of this section is to add to our understanding of computable queries by investigating an equivalence relation on the set CQ of computable queries; two such queries are related if they are realised via the same Turing-computable mapping, say δ. In particular, we are interested in the cardinality of the equivalence class of a computable query, realised via δ and some encoding, with respect to the said equivalence relation; this equivalence class is denoted by Δ_δ. In the case when $|\Delta_\delta| = 0$, there is no computable query τ such that τ is realised via δ and some encoding. On the other hand, when $|\Delta_\delta| = 1$ and τ is realised via δ and all possible encodings in Π, then τ corresponds to an encoding-independent computable query. We show that both of the decision problems: is $|\Delta_\delta| = 0$? and is $|\Delta_\delta| = 1$? are undecidable. We also show the interesting result that if $|\Delta_\delta| > 1$, then $|\Delta_\delta| = \omega$. Our interpretation of this result is that it is not possible to obtain a finer partition of the class of computable queries CQ induced by Δ_δ, without putting restrictions on the set of encodings (for example, by only allowing free encodings).

We next formally define the equivalence class Δ_δ.

Definition 5.1. Let $\delta \in$ TC and $S \in \Psi$. Δ_δ is defined by $\Delta_\delta = \{\tau \mid \exists\, \rho \in \Pi$ such that τ is realised via δ and $\rho\}$ and $\Delta_\delta(S)$ is defined by $\Delta_\delta(S) = \{\tau(S) \mid \tau \in \Delta_\delta\}$.

The following proposition gives us yet another characterisation of an encoding-independent computable query.

Proposition 5.1. A computable query $\tau \in \Delta_\delta$ is encoding-independent if and only if $|\Delta_\delta| = 1$ and $\{\rho \mid \tau \in \Delta_\delta$ and τ is realised via δ and $\rho\} = \Pi$. $\square$

(See Definition 3.2 in order to understand why both conditions in the statement of the above proposition are necessary.)

The following lemma shows the undecidability of whether $|\Delta_\delta| = 0$ and whether $|\Delta_\delta| = 1$.

Lemma 5.2. The two decision problems,

(1) is $|\Delta_\delta| = 0$?;

(2) is $|\Delta_\delta| = 1$?

are both undecidable.

Proof. In order to prove the result we utilise Rice's theorem [RICE53]. In particular, we need to show that the sets, CL_0 and CL_1, of Turing-computable mappings, δ, such that $|\Delta_\delta| = 0$ and $|\Delta_\delta| = 1$, respectively, are both non-trivial. That is, $CL_i \neq \emptyset$ and $CL_i \neq$ TC, for $i = 0,1$. We prove the result by exhibiting $\delta_0, \delta_1 \in$ TC, such that $\delta_0 \in CL_0$ and $\delta_1 \in CL_1$.

Let δ_0 be a Turing-computable mapping that removes all the delimiters in Delim from its input and then halts. It can easily be verified that $|\Delta_{\delta_0}| = 0$, since $\forall\, \rho \in \Pi$, $\forall\, S \in \Psi$, $\delta_0(ENC(\rho, S))$ cannot be decoded by using ρ^{-1}.

Let δ_1 be a Turing-computable mapping that returns [[]] (i.e. it returns the string representing the singleton containing the empty record) if its input is an encoding of a set containing an even number of records, and returns [] (i.e. it returns the string representing the empty set of records) otherwise. It can easily be verified that $|\Delta_{\delta_1}| = 1$. □

The following lemma shows the undecidability of whether, $\forall$ k $\in \omega$, $|\Delta_\delta| > k$.

Lemma 5.3. $\forall$ k $\in \omega$, the decision problem, is $|\Delta_\delta| > k$?, is undecidable.

Proof. Again we utilise Rice's theorem [RICE53] to show that $\forall$ k $\in \omega$, the set $CL_{>k}$ of Turing-computable mappings δ such that $|\Delta_\delta| > k$ is non-trivial. That is, $\forall$ k $\in \omega$, $CL_{>k} \neq \emptyset$ and $CL_{>k} \neq$ TC.

Let δ be a Turing-computable mapping that returns [[]] (i.e. it returns the string representing the singleton containing the empty record) if its encoded input does not contain exactly k records, otherwise it returns the encoded list containing only the first record in its encoded input. It can easily be verified that $|\Delta_\delta| > k$, since whenever $S \in \Psi$ and $|S| > k$, $\exists\, \rho_1, ..., \exists\, \rho_{k+1}$ such that $\forall$ i,j $\in$ {1,...,k+1}, $i \neq j$, $DEC(\rho_i^{-1}, \delta(ENC(\rho_i, S))) \neq DEC(\rho_j^{-1}, \delta(ENC(\rho_j, S)))$. The result now follows, since $\forall$ k $\in \omega$, $CL_{>k}$ is non-trivial, on using Lemma 5.2. □

The following theorem shows the interesting result that for no n > 1, n $\in \omega$, does there exist a $\delta \in$ TC such that $|\Delta_\delta| = n$. That is, either $|\Delta_\delta| = 0$ or $|\Delta_\delta| = 1$ or $|\Delta_\delta| = \omega$. In the proof of this theorem we use the expression "$\rho_i, \rho_j \in \Pi$ *agree* on $S \in \Psi$ with respect to τ", where τ is realised via δ, to mean that $\tau(S) = DEC(\rho_i^{-1}, \delta(ENC(\rho_i, S))) = DEC(\rho_j^{-1}, \delta(ENC(\rho_j, S)))$.

Theorem 5.4. If $|\Delta_\delta| > 1$, then $|\Delta_\delta| = \omega$.

Proof. Suppose that $\exists$ n > 1, n $\in \omega$, such that $|\Delta_\delta| = n$. Then $\exists\, \rho_1, ..., \rho_n \in \Pi$ such that $\forall$ i $\in$ I, τ_i is realised via δ and ρ_i, and $\forall$ i, j $\in$ I, where $i \neq j$, $\tau_i \neq \tau_j$. We obtain a contradiction by a diagonalisation argument that constructs a computable query $\tau_{n+1} \in \Delta_\delta$ such that τ_{n+1} is realised via δ and ρ_{n+1} and $\forall$ i $\in$ I, $\tau_{n+1} \neq \tau_i$.

By our assumption that $|\Delta_\delta| = n$, $\exists\, S_1, ..., S_{n-1} \in \Psi$ such that $\forall$ i $\in$ {1,...,n–1}, $\tau_i(S_i) \neq \tau_{i+1}(S_i)$. We note that we can assume that both $\tau_i(S_i)$ and $\tau_{i+1}(S_i)$ are defined, since if one of them is undefined, say $\tau_i(S_i)$, then we can assign to it a defined output not equal to $\tau_{i+1}(S_i)$ without affecting the argument. Now, without loss of generality, we assume that $\forall$ i, j $\in$ {1,...,n–1}, $i \neq j$, $S_i \neq S_j$ and $adom(S_i) \cap adom(S_j) = \emptyset$. We note that indeed there is no loss of generality in this assumption, since if $\exists$ i, j $\in$ {1,...,n–1}, $i \neq j$, such that $S_i = S_j$ or $adom(S_i) \cap adom(S_j) \neq \emptyset$, then we can choose $S_j^{\equiv} \in \Psi$ instead of S_j such that $S_j^{\equiv}$ is isomorphic to S_j, $S_j^{\equiv} \neq S_j$ and $adom(S_j^{\equiv}) \cap adom(S_j) = \emptyset$. Furthermore, we can let ρ_k agree with itself on S_j and $S_j^{\equiv}$ with respect to τ_k, for k = j, j+1, without affecting the argument. So, $\tau_j(S_j^{\equiv}) \neq \tau_{j+1}(S_j^{\equiv})$ would follow as required.

We now construct $\rho_{n+1} \in \Pi$ such that $\forall$ i $\in$ {1,...,n–1}, ρ_{n+1} agrees with ρ_i on S_i with respect to τ_i. That is, $\forall$ i $\in$ {1,...,n–1}, $\tau_{n+1}(S_i) = \tau_i(S_i)$. It therefore follows that $\forall$ i $\in$ {1,...,n–1}, $\tau_{n+1}(S_i) \neq \tau_{i+1}(S_i)$. It therefore remains to show that $\exists\, S_n \in \Psi$ such that $\tau_{n+1}(S_n) \neq \tau_1(S_n)$.

Let $S_n \in \Psi$ be isomorphic to S_1 and such that $\forall$ i $\in$ {1,...,n−1}, $S_n \neq S_i$ and $adom(S_n) \cap adom(S_i) = \emptyset$. We now assume without loss of generality that $\tau_1(S_1) = \tau_1(S_n)$ and that $\tau_2(S_1) = \tau_2(S_n)$. We note that indeed there is no loss of generality in this assumption, since we can let ρ_k agree with itself on S_1 and S_n with respect to τ_k for k = 1,2, without affecting the argument. It follows that $\tau_1(S_n) \neq \tau_2(S_n)$, since $\tau_1(S_1) \neq \tau_2(S_1)$. We thus augment the construction of ρ_{n+1} by letting it agree with ρ_2 on S_n with respect to τ_2, implying that $\tau_{n+1}(S_n) = \tau_2(S_n)$. The result that $\tau_{n+1}(S_n) \neq \tau_1(S_n)$ now follows as required. □

The following proposition shows that it is possible for a Turing-computable mapping δ to give rise to a countably infinite number of different answers for a given input, each such answer corresponding to a different computable query that δ realises.

Proposition 5.5. $\exists\, \delta \in$ TC, $\exists\, S \in \Psi$ such that $|\Delta_\delta(S)| = \omega$.

Proof. Let δ be the Turing-computable mapping that inserts the encoded record [("A", "a")] into its encoded input and let S = ∅. Then the isomorphism θ of a mapping $\rho = \theta \circ \phi \in \Pi$ can map "A" to any attribute in U and correspondingly it can map "a" to any value in D. Thus there exist ω computable queries τ in Δ_δ; the result τ(S) of each such query is to insert a different record with a single attribute and its associated value into its input S. □

The following proposition shows that it is not necessarily the case that for all encodings $\rho \in \Pi$ there exists a computable query $\tau \in \Delta_\delta$ such that τ is realised via δ and ρ. This justifies the result of Proposition 5.1.

Proposition 5.6. $\exists\, \delta \in$ TC such that $\{\rho \mid \exists\, \tau \in \Delta_\delta$ and τ is realised via δ and ρ$\} \neq \Pi$.

Proof. Assume without loss of generality that "*" ∈ Σ and recall that Delim ⊂ Σ, where Delim = {[,], (,),,}. Consider the following.

(1) Let δ be the Turing-computable mapping that outputs the encoded list containing only the first record in its encoded input if the number of records in its input is at least 1, otherwise it outputs [] (i.e. it outputs the string representing the empty set of records); it therefore follows that $|\Delta_\delta| > 1$.

(2) Alternatively, let δ be the Turing-computable mapping that outputs [[]] (i.e. it outputs the string representing the singleton containing the empty record) if the number of records in its input is n, otherwise it outputs [] (i.e the string representing the empty set of records); it therefore follows that $|\Delta_\delta| = 1$.

In both (1) and (2) we specify an exception condition for δ, namely when the first symbol in its input which is not in Delim is equal to "*". Whenever this exception condition is true, δ removes all the delimiters in Delim from its input and then halts. It follows that, in this case, τ is realised via δ and ρ if and only if $\rho = \theta \circ \phi \in \Pi$ and $\nexists$ A ∈ U such that θ(A) = "*". □

6. Enumerable Queries

In this section we investigate enumerable queries and show the equivalence between the class of *generic enumerable queries* [ABIT90] and a subclass of enumerable queries, namely the class of encoding-independent enumerable queries. We define an enumerable query R to be a relation between sets that can be enumerated via a Turing-enumerable relation T, and an encoding ρ, which encodes ordered pairs in R. We show that all the graphs of encoding-independent computable queries are encoding-independent enumerable queries but that there exists an encoding-independent enumerable query, say R, such that no subset of R, which has the same domain as R, is the graph of a computable query which is encoding-independent.

Informally, an enumerable query is a relation $R \subseteq \Psi \times \Psi$ that can be enumerated via a Turing-enumerable relation $T \subseteq \Sigma^* \times \Sigma^*$ and an ed-oracle which encodes ordered pairs in R by using an encoding $\rho \in \Pi$.

Definition 6.1. Let (ENC, DEC) be an ed-oracle. A relation $R \in \Psi \times \Psi$ is an *enumerable query* if

$$\exists\, T \in TE, \exists\, \rho \in \Pi, \forall\, (S_1, S_2) \in \Psi \times \Psi,$$
$$(ENC(\rho, S_1), ENC(\rho, S_2)) \in T \text{ if and only if } (S_1, S_2) \in R.$$

As an abbreviation to the above statement we say that R is *enumerated* via T and ρ; at times we simply say that R is enumerated via T meaning that $\exists\, \rho \in \Pi$, such that R is realised via T and ρ. Hereafter, we let EQ denote the set of all enumerable queries.

The following lemma shows that the graph of a mapping in Φ is a computable query if and only if it is an enumerable query.

Lemma 6.1. Let $\tau \in \Phi$. Then $\tau \in CQ$ if and only if $graph(\tau) \in EQ$.

Proof. (*If*): Assume $graph(\tau) \in EQ$ and that $graph(\tau)$ is enumerated via T and ρ. Then $\exists\, \delta \in TC$ such that $graph(\delta) = T$ [ROGE67], i.e. δ is a Turing-computable mapping. It remains to show that $\forall\, S \in \Psi$, $\tau(S) = DEC(\rho^{-1}, \delta(ENC(\rho, S)))$. Now, since $graph(\tau)$ is enumerated via T and ρ, we have that $(S, \tau(S)) \in graph(\tau)$ if and only if $(ENC(\rho, S), \delta(ENC(\rho, S))) \in graph(\delta)$ or equivalently that $ENC(\rho, \tau(S)) = \delta(ENC(\rho, S))$. The result follows.

(*Only if*): Assume that τ is realised via δ and ρ. Then $graph(\delta) \in TE$ [ROGE67], i.e. $graph(\delta)$ is a Turing-enumerable relation. It remains to show that $\forall\, S \in \Psi$, $(S, \tau(S)) \in graph(\tau)$ if and only if $(ENC(\rho, S), \delta(ENC(\rho, S))) \in graph(\delta)$. Now, since τ is realised via δ and ρ, we have that $\tau(S) = DEC(\rho^{-1}, \delta(ENC(\rho, S)))$ or equivalently that $(S, DEC(\rho^{-1}, \delta(ENC(\rho, S)))) \in graph(\tau)$. The result follows. □

The following lemma shows that for each enumerable query, say R, there exists a computable query having the same domain as R and such that its graph is contained in R.

Lemma 6.2. Let $R \in EQ$. Then $\exists\, \tau \in CQ$ such that $graph(\tau) \subseteq R$ and the domain of τ is equal to the domain of R.

Proof. Assume that R is enumerated via T and ρ. Then by the single-valuedness theorem [ROGE67], $\exists\, \delta \in TC$ such that $graph(\delta) \subseteq T$ and such that the domain of δ is equal to the domain of T. Let $graph(\tau)$ be such that $(S_1, S_2) \in graph(\tau)$ if and only if $(ENC(\rho, S_1), ENC(\rho, S_2)) \in graph(\delta)$. It follows that the domain of τ is equal to the domain of R. Finally, $S_2 = \tau(S_1) =$

$DEC(\rho^{-1}, \delta(ENC(\rho, S_1)))$ as required, since $ENC(\rho, S_2) = \delta(ENC(\rho, S_1))$. □

We next investigate the relationship between Definition 6.1 of an enumerable query and the definition of a *generic enumerable query* as given in [ABIT90]. (We note that in [ABIT90] a generic enumerable query is called a non-deterministic database transformation.) Informally, an enumerable query, say R, is generic if an ordered pair, say (S_1, S_2), is enumerated by R if and only if every ordered pair which is isomorphic to (S_1, S_2) is also enumerated by R, i.e. R is closed under isomorphism. The formal definition of a generic enumerable query, recast in our context and notation, follows.

Definition 6.2. A relation $R \in \Psi \times \Psi$ is a *generic enumerable query* if

(1) R is an *order-independent* enumerable query (see Definition 6.4), and

(2) R is *generic*, i.e. for all isomorphisms $\eta \in \Phi$,

$(S_1, S_2) \in R$ if and only if $(\eta(S_1), \eta(S_2)) \in R$.

We note that condition (1) of Definition 6.2 is stated in [ABIT90] as: R is a recursively enumerable relation. We call our interpretation of this statement, as given in (1) of Definition 6.2, the *order-independence assumption*. We further note that in [ABIT90] enumerable queries are considered to be C-generic, i.e. the constants explicitly named in the query are mapped by the isomorphism η to themselves (i.e. the isomorphism is the identity mapping on C). As with Definition 3.1 of a generic query, for simplicity, we assume that C is the empty set.

Informally, an enumerable query, say R, is encoding-independent if its enumeration does not depend on any particular encoding $\rho \in \Pi$ used by an ed-oracle.

Definition 6.3. An enumerable query $R \in EQ$ is *encoding-independent* if

$\exists\, T \in TE, \forall \rho \in \Pi, \forall (S_1, S_2) \in \Psi \times \Psi,$

$(ENC(\rho, S_1), ENC(\rho, S_2)) \in T$ if and only if $(S_1, S_2) \in R$.

Hereafter, we let EIEQ denote the set of all encoding-independent enumerable queries. The following lemma, which follows directly from Lemma 6.1 and Definitions 3.2 and 6.3, shows that the graph of a mapping $\tau \in \Phi$ is an encoding-independent enumerable query if and only if τ is an encoding-independent computable query.

Lemma 6.3. Let $\tau \in \Phi$. Then $\tau \in EI$ if and only if $graph(\tau) \in EIEQ$. □

Informally, an enumerable query is order-independent if its enumeration does not depend on any ordering function ϕ of an encoding $\rho = \theta \circ \phi \in \Pi$ used by an ed-oracle.

Definition 6.4. We define a Turing-enumerable relation $T \in TE$ to be *order-independent* if

$\forall\, \theta_i \circ \phi_i \in \Pi, \forall \rho = \theta \circ \phi \in \Pi$ with $\theta = \theta_i, \forall (S_1, S_2) \in \Psi \times \Psi,$

$(ENC(\theta \circ \phi_i, S_1), ENC(\theta \circ \phi_i, S_2)) \in T$ if and only if $(ENC(\rho, S_1), ENC(\rho, S_2)) \in T$.

Hereafter, we let OITE denote the set of all order-independent Turing-enumerable relations. An enumerable query $R \in EQ$ is *order-independent* if $\exists\, T \in$ OITE such that R is enumerated via T.

(Recall the order-independence assumption made after Definition 6.2, which stated that a generic enumerable query is order-independent.)

The following theorem shows that the class of generic enumerable queries is equivalent to the class of encoding-independent queries, thus extending the result of Theorem 3.1 and providing us with an alternative definition to the standard definition of a generic enumerable query given in [ABIT90].

Theorem 6.4. An enumerable query $R \in EQ$ is encoding-independent if and only if it is a generic enumerable query.

Proof. Let $\eta \in \Phi$ be an isomorphism, $(S_1, S_2) \in \Psi \times \Psi$ and assume that R is enumerated via T $\in$ OITE and an encoding $\rho \in \Pi$.

(*If*): Without loss of generality let $\eta = \rho_1^{-1} \circ \rho_2$, where $\rho_1 = \theta_1 \circ \phi_1$, $\rho_2 = \theta_2 \circ \phi_2 \in \Pi$. Also assume that $\rho = \rho_1$. It remains to show that $(ENC(\rho_1, S_1), ENC(\rho_1, S_2)) \in T$ if and only if $(ENC(\rho_2, S_1), ENC(\rho_2, S_2)) \in T$.

Now, suppose that $(ENC(\rho_1, S_1), ENC(\rho_1, S_2)) \in T$ and thus $(S_1, S_2) \in R$. On using genericity we have that $(\eta(S_1), \eta(S_2)) \in R$ and thus $(ENC(\rho_1, \eta(S_1)), ENC(\rho_1, \eta(S_2))) \in T$. Thus we also have that $(ENC(\rho_1 \circ \eta, S_1), ENC(\rho_1 \circ \eta, S_2)) \in T$, since $\rho_1 \circ \eta$ is also an encoding in Π. It follows that $(ENC(\theta_2 \circ \phi, S_1), ENC(\theta_2 \circ \phi, S_2)) \in T$ for some ordering function ϕ, on using Lemma 2.2. Thus by the order-independence assumption we obtain $(ENC(\theta_2 \circ \phi_2, S_1), ENC(\theta_2 \circ \phi_2, S_2)) \in T$, which is equivalent to $(ENC(\rho_2, S_1), ENC(\rho_2, S_2)) \in T$, as required.

Now suppose that $(ENC(\rho_2, S_1), ENC(\rho_2, S_2)) \in T$ and thus $(S_1, S_2) \in R$. On using genericity we have that $(\eta^{-1}(S_1), \eta^{-1}(S_2)) \in R$ and thus $(ENC(\rho_2, \eta^{-1}(S_1)), ENC(\rho_2, \eta^{-1}(S_2))) \in T$. Thus we also have that $(ENC(\rho_2 \circ \eta^{-1}, S_1), ENC(\rho_2 \circ \eta^{-1}, S_2)) \in T$, since $\rho_2 \circ \eta^{-1}$ is also an encoding in Π. On using an argument similar to that used in Lemma 2.2 it follows that $(ENC(\theta_1 \circ \phi, S_1), ENC(\theta_1 \circ \phi, S_2)) \in T$, for some ordering function ϕ. Thus by the order-independence assumption it follows that $(ENC(\theta_1 \circ \phi_1, S_1), ENC(\theta_1 \circ \phi_1, S_2)) \in T$, which is equivalent to $(ENC(\rho_1, S_1), ENC(\rho_1, S_2)) \in T$ as required.

(*Only if*): By Definitions 6.3 and 6.4 it follows that R is order-independent. It remains to show that R is a generic enumerable query.

By Definition 6.3 of encoding-independence we have that $(ENC(\rho, S_1), ENC(\rho, S_2)) \in T$ or equivalently $(ENC(\rho \circ \eta^{-1}, S_1), ENC(\rho \circ \eta^{-1}, S_2)) \in T$ if and only if $(S_1, S_2) \in R$, since $\rho \circ \eta^{-1}$ is also an encoding in Π. Thus $(ENC(\rho \circ \eta^{-1}, \eta(S_1)), ENC(\rho \circ \eta^{-1}, \eta(S_2))) \in T$ if and only if $(\eta(S_1), \eta(S_2)) \in R$, since R is an enumerable query. It therefore follows that $(ENC(\rho \circ \eta^{-1} \circ \eta, S_1), ENC(\rho \circ \eta^{-1} \circ \eta, S_2)) \in T$ if and only if $(\eta(S_1), \eta(S_2)) \in R$, since again $\rho \circ \eta^{-1} \circ \eta$ is also an encoding in Π. The result now follows, i.e. $(ENC(\rho, S_1), ENC(\rho, S_2)) \in T$ if and only if $(\eta(S_1), \eta(S_2)) \in R$, since $\eta^{-1} \circ \eta$ is the identity mapping on Ψ. □

The following lemma shows that the result of Lemma 6.2 cannot be generalised to encoding-independent computable queries.

Lemma 6.5. $\exists\, R \in$ EIEQ such that for no $\tau \in$ CQ satisfying graph(τ) $\subseteq$ R, with the domain of τ equal to the domain of R, is it true that $\tau \in$ EI, i.e. for no such computable query τ is τ encoding-independent.

Proof. Let (S, {t}) $\in$ R if and only if t $\in$ S, i.e. R is the set membership relation. It can easily be verified that R $\in$ EIEQ. Now, by Lemma 6.2, $\exists\, \tau \in$ CQ such that graph(τ) $\subseteq$ R and the domain of τ is equal to the domain of R. Thus graph(τ) $\in$ EQ, by Lemma 6.1. In order to conclude the proof we demonstrate that graph(τ) $\notin$ EIEQ which implies, by Lemma 6.3, that $\tau \notin$ EI as required.

Assume to the contrary that graph(τ) $\in$ EIEQ. Let S = $\{t_1, t_2\}$ and t = t_1, assuming without loss of generality that $|t_1| = |t_2|$, that adom($\{t_1\}$) $\cap$ adom($\{t_2\}$) = $\emptyset$ and that $\eta \in \Phi$ is an automorphism on S such that $\eta(\{t_1\}) = \{t_2\}$ and $\eta(\{t_2\}) = \{t_1\}$. It follows that $(\{t_1, t_2\}, \{t_2\}) \notin$ graph(τ), since τ is a mapping. Now, by Theorem 6.4, graph(τ) is a generic enumerable query and thus $(\{t_1, t_2\}, \{t_1\}) \in$ graph(τ) if and only if $(\eta(\{t_1, t_2\}), \eta(\{t_1\})) \in$ graph(τ). It therefore follows that $(\{t_1, t_2\}, \{t_2\}) \in$ graph(τ). This leads to a contradiction, so graph(τ) $\notin$ EIEQ. □

The following lemma exhibits a special case when an encoding-independent computable query can be obtained from an encoding-independent enumerable query. In particular, the lemma applies to the set membership relation utilised in the proof of Lemma 6.5, since all the sets S $\in \Psi$ are finite.

Lemma 6.6. Let R $\in$ EIEQ and let graph(τ) = $\{(S_1, S_2) \mid \exists\, (S_1, S_3) \in R$ and $S_2 = \cup \{S_3 \mid (S_1, S_3) \in R\}\}$. Then $\tau \in$ EI, if $\forall\, (S_1, S_2) \in$ graph(τ), $|S_2| \in \omega$, i.e. the cardinality of S_2 is finite.

Proof. The result follows by Lemma 6.3, since graph(τ) $\in$ EIEQ and $\tau \in \Phi$. □

We conclude this section by noting that similar results to those obtained in Section 4 can be obtained by defining the corresponding classes of free-encoding enumerable queries and C-encoding-independent enumerable queries.

7. Arbitrary-Encoding Computable Queries

In this section we define another class of computable queries, namely the class of arbitrary-encoding computable queries, which are computable queries obtained by fixing the encoding used to realise a given computable query. For example, the fixed encoding may correspond to the physical layout of the database records. In fact, we show that the class of arbitrary-encoding computable queries is equivalent to the class CQ of all computable queries and thus there is no loss of computational expressiveness when restricting ourselves to a single fixed encoding. We then define four arbitrary-encoding computable queries, called *membership queries*. The first two membership queries return the first record in a set and the first pair in a record, respectively, given the ordering function of the fixed encoding; the other two membership queries delete the first record in a set and the first pair in a record, respectively, again given the ordering function of the fixed encoding. We show that, by adding membership queries to the class of encoding-independent computable queries, the closure of the resulting extended class under composition of mappings is the set II of all isomorphism-independent computable queries. In this way we can enhance a database language, which is computationally complete with respect to the class EI, with membership queries by sacrificing order-independence but without sacrificing isomorphism-independence.

The following assumption derives from the axiom of choice in set theory [HALM74].

The Assumption of Choice of Encoding. There exists a mapping from the singleton $\{\Pi\}$ into Π, which we denote by *choice*, such that *choice*$(\Pi) \in \Pi$.

As a result of the above assumption, which will be made from now on, we will refer to *choice* as a given constant mapping. In the following we abbreviate *choice*(Π) to *choice*.

We now define the notion of an arbitrary-encoding computable query with respect to *choice*.

Definition 7.1. A computable query $\tau \in$ CQ is an *arbitrary-encoding computable query* with respect to *choice* (or simply an arbitrary-encoding computable query whenever *choice* is understood from context) if

$$\exists\, \delta \in \mathrm{TC}, \forall\, S \in \Psi, \tau(S) = \mathrm{DEC}(\mathit{choice}^{-1}, \delta(\mathrm{ENC}(\mathit{choice}, S))).$$

That is, $\exists\, \delta \in$ TC such that τ is realised via δ and *choice*. Hereafter, we let AE(*choice*) denote the set of all arbitrary-encoding computable queries with respect to *choice* (or simply AE whenever *choice* is understood from context). An immediate consequence of Definition 7.1 and Proposition 2.4 is that AE = CQ, i.e. we can restrict ourselves to a single encoding without any loss of computational expressive power.

We now briefly consider the application of the class of arbitrary-encoding computable queries, AE, with a view to enhancing a database language, which is computationally complete with respect to the class EI, such as the database language QL [CHAN80].

Let *choice* $= \rho = \theta \circ \phi$, $S \in \Psi$ and $t \in S$. We define two computable queries, which can easily be shown to be in AE(*choice*):

(1) set_member(S) returns a singleton containing the first record in $\phi(S)$ if $S \neq \emptyset$; otherwise $\emptyset$ is returned;

(2) record_member({t}) returns a singleton record containing the first ordered pair in $\phi(\{t\})$ if $t \neq \emptyset$; otherwise {<>} is returned.

We next define two further computable queries, which can also easily be shown to be in AE(*choice*):

(3) delete_record(S) returns S – set_member(S);

(4) delete_pair({t}) returns {t – record_member({t})}.

As mentioned earlier, we refer to queries (1), (2), (3) and (4) as *membership queries* and designate the set comprising these queries by MEM. It is obvious that MEM $\not\subseteq$ EI and that MEM $\subseteq$ II. On the other hand, it can be seen that MEM $\not\subseteq$ OI, since the definition of MEM is dependent upon the ordering function ϕ of *choice*. In addition, it can easily be verified that by repetitive application of the membership queries the ordering resulting from the ordering function ϕ of *choice* can be simulated. We claim that membership queries are useful in a database language, since they allow us to simulate queries which are not in OI whenever this is deemed to be desirable.

Let EI+MEM denote the class of computable queries EI $\cup$ MEM closed under composition of mappings. The next theorem shows that the expressive power of EI+MEM is exactly that of the set of all isomorphism-independent computable queries in AE $\cap$ II. This implies that EI+MEM = II, since AE $\cap$ II = II follows from the fact that AE = CQ.

Theorem 7.1. Let *choice* = $\rho = \theta \circ \phi$ and let $\tau \in$ AE be realised via δ. Assume that $\{\rho_i \mid \exists\, \tau_i \in \Delta_\delta$ and τ_i is realised via δ and $\rho_i\} = \Pi$ and that $\forall$ S $\in \Psi$, $\tau(S)$ is defined if and only if $\forall\, \tau_i \in \Delta_\delta$, $\tau_i(S)$ is defined. Then $\tau \in$ II if and only if $\tau \in$ EI+MEM.

Proof. (*If*): Let $\tau \in$ EI+MEM. We prove the result by induction on the number of compositions, say k, of mappings in EI $\cup$ MEM used to obtain τ.

(*Basis*): If k = 0, then $\tau \in$ EI $\cup$ MEM implies that τ is realised via δ and ρ, and thus $\tau \in$ II by definition.

(*Induction*): Assume the result holds for k compositions, we then need to prove that the result holds for k+1 compositions. Now, suppose that k+1 compositions were used in order to obtain τ. Then $\tau = \tau_1 \circ \tau_2 \circ ... \circ \tau_k \circ \tau_{k+1}$ and by inductive hypothesis $\tau_0 = \tau_2 \circ ... \circ \tau_k \circ \tau_{k+1}$ is in II and so is τ_1. It remains to show that $\tau_1 \circ \tau_0 \in$ II.

We assume that τ_0 is realised via δ_0 and ρ and that τ_1 is realised via δ_1 and ρ, since both $\tau_0, \tau_1 \in$ II. Thus $\forall$ S $\in \Psi$, $\tau(S) = (\tau_1 \circ \tau_0)(S) = \tau_1(\tau_0(S)) = \mathrm{DEC}(\rho^{-1}, \delta_1(\mathrm{ENC}(\rho, S_1)))$, where $S_1 = \mathrm{DEC}(\rho^{-1}, \delta_0(\mathrm{ENC}(\rho, S)))$. It follows that $\tau(S) = \mathrm{DEC}(\rho^{-1}, \delta_1 \circ \rho \circ \rho^{-1} \circ \delta_0(\mathrm{ENC}(\rho, S)))$. Therefore, by Proposition 2.4, we have that τ is realised via $\delta = \rho \circ \rho^{-1} \circ \delta_1 \circ \rho \circ \rho^{-1} \circ \delta_0 = \delta_1 \circ \rho \circ \rho^{-1} \circ \delta_0$ and ρ and thus $\tau \in$ II as required.

(*Only if*): Let $R = \cup_i \{\mathrm{graph}(\tau_i) \mid \tau_i \in \Delta_\delta\}$. It follows that R $\in$ EIEQ, since Turing-enumerable relations are closed under union [ROGE67] and $\{\rho_i \mid \exists\, \tau_i \in \Delta_\delta$ and τ_i is realised via δ and $\rho_i\} = \Pi$. Furthermore, let $\mathrm{graph}(\alpha) = \{(S_1, S_2) \mid \exists\, (S_1, S_3) \in R \text{ and } S_2 = \cup \{S_3 \mid (S_1, S_3) \in R\}\} = \cup_i \{\tau_i(S_1) \mid \tau_i \in \Delta_\delta\}$. We claim that $\forall\, (S_1, S_2) \in \mathrm{graph}(\alpha)$, $|S_2| \in \omega$, i.e. the cardinality of S_2 is finite. This claim is now proved; by assumption $\tau \in$ II and there are only a finite number of orderings of the set $S_1 \in \Psi$. This implies that the set S_2 is indeed finite, whenever $(S_1, S_2) \in \mathrm{graph}(\alpha)$, since in this case $\forall\, \tau_i \in \Delta_\delta$, $\tau_i(S_1)$ is defined. Thus, by Lemma 6.6, $\alpha \in$ EI.

We observe that $\forall\, S_1 \in \Psi$, $\tau(S_1) \subseteq S_2$, whenever $\tau(S_1)$ is defined. Thus in order to conclude the proof we show how we can utilise membership queries so as to test for each $t \in S_2$ whether $t \in \tau(S_1)$ holds. First we show how to test whether $t \in \cap_i\, \tau_i(S_1)$ holds. This test can be computed by a query in EI+MEM by setting $S_2 = \cap_i\, \tau_i(S_1)$ in the construction of α above, instead of setting $S_2 = \cup_i\, \tau_i(S_1)$; it can then be verified that $\alpha \in$ EI would also follow.

Second we show how we can test for each $t \in S_2 - (\cap_i \tau_i(S_1))$, whether $t \in \tau(S_1)$ holds. We observe that in this case if $t \in \tau(S_1)$ holds, then t must have been generated as a result of the ordering induced by the ordering function ϕ when encoding S_1.

Let $t_1, t_2 \in S_1$, $(A_1, v_1), (A_2, v_2) \in t_1$ and let $n > 0$ denote a natural number local to the proof. We say that $t_1 <_n t_2$ holds in S_1, if the single element of the list $\phi(\{t_1\})$ appears exactly n positions before the single element of the list $\phi(\{t_2\})$ in the list $\phi(S_1)$. Correspondingly, we say that $(A_1, v_1) <_n (A_2, v_2)$ holds in t_1, if (A_1, v_1) appears exactly n positions before (A_2, v_2) in the single element of the list $\phi(\{t_1\})$. Thus, where m denotes a natural number again local to the proof, $t \in \tau(S_1)$ holds if and only if there is a sequence,

$$u_0 <_{n_1} u_1 , ..., u_{m-1} <_{n_m} u_m, \tag{1}$$

such that for each $j \in J$, $u_{j-1} <_{n_j} u_j$ holds in either S_1 or in a particular record $t_k \in S_1$.

Let Qs^n denote the mapping that returns a singleton containing the nth record $t \in S$ if $|S| \geq n$, otherwise it returns $\emptyset$, and let Qr^n denote the mapping that returns a singleton containing the nth ordered pair $(A, v) \in t$ if $|t| \geq n$, otherwise it returns $\{<>\}$. It can now be verified by straightforward induction on n that both the mappings Qs^n and Qr^n are in EI+MEM. We note that the induction step is proved by applying delete_record n−1 times on S and then applying set_member once on the result, and by applying delete_pair n−1 times on t and then applying record_member once on the result, respectively. It follows that, for $n = 1$, $Qs^n(S)$ = set_member(S) and $Qr^n(t)$ = record_member(t), and, for $n > 1$, $Qs^n(S) = \{t\}$ if and only if $Qs^1(S) = \{t_1\}$ and $t_1 <_{n-1} t$ holds in S, and $Qr^n(\{t\}) = \{(A, v)\}$ if and only if $Qr^1(\{t\}) = \{(A_1, v_1)\}$ and $(A_1, v_1) <_{n-1} (A, v)$ holds in t.

Let us call queries of the form Qs^n and Qr^n *order-dependent* queries. It follows that by repetitive application of such queries the test $u_{j-1} <_{n_j} u_j$, where $j \in J$, can be simulated in EI+MEM. Furthermore, it follows that any sequence of such tests, such as equation (1) given above, can also be simulated in EI+MEM as a composition of mappings.

We finally show that the test, $t \in \tau(S_1)$, where $t \in S_2 - (\cap_i \tau_i(S_1))$, can be computed by a query in EI+MEM, which utilises one or more order-dependent queries. Now, by the definition of t, we can assume without loss of generality that either $\exists\, \tau_i \in \Delta_\delta$, with $\tau_i \neq \tau$, such that $t \in \tau_i(S_1)$ or $t \in \tau(S_1)$ but t is not a member of both, where τ_i is realised via δ and $\theta \circ \phi_i$ with $\phi_i \neq \phi$. We next prove that $t \in \tau(S_1)$ holds if and only if $\beta(S_1) = \{t\}$, where $\beta \in$ EI+MEM.

The query β simulates the computation of δ on $ENC(\rho, S_1)$ and returns $\{t\}$ if and only if the test $t \in \tau(S_1)$ holds. Thus $\beta = \beta_1 \circ \beta_2 \circ ... \circ \beta_k$ is a composition of mappings where each β_q, for $q = 1,...,k$, is either in EI or is an order-dependent query. The result now follows, since EI+MEM is closed under composition of mappings. □

Finally, we note that the class EIFE can be enhanced with membership queries in a similar fashion. We further note that in this case only set_member and record_member are required to be membership queries, since both delete_record and delete_pair can be replaced by computable queries in EIFE. In particular, delete_record(S) can be replaced by the computable query

$delete\_record_S(\{t\})$, which returns $S - \{t\}$, where $\{t\}$ can be obtained by applying set_member(S), and delete_pair({t}) can be replaced by the computable query, $delete\_pair_{\{t\}}(\{(A, v)\})$, which returns $\{t - \{(A, v)\}\}$, where $\{(A, v)\}$ can be obtained by applying record_member({t}).

8. Concluding Remarks

We have presented an alternative approach to that of Chandra and Harel [CHAN80] and Abiteboul and Vianu [ABIT90] in considering computable queries, i.e. as being realised via a Turing-computable mapping and an encoding. We have shown the equivalence of the class of encoding-independent computable queries and the class of generic computable queries [CHAN80]. In addition, we have shown the equivalence of the class of encoding-independent enumerable queries and the class of generic enumerable queries [ABIT90]. These equivalences provide us with a better insight into the notion of a computable database query. We have also shown that the class of free-encoding-independent computable queries is a very natural class to consider, since it corresponds to the class of all C-generic computable queries; both in [ABIT90] and [HULL93] a database language is considered to be computationally complete if it expresses exactly this class of queries.

We further investigated an equivalence relation on the set CQ of computable queries; two such queries are related if they are realised via the same Turing-computable mapping, δ. We have shown that either $|\Delta_\delta| = 0$, or $|\Delta_\delta| = 1$, or $|\Delta_\delta| = \omega$. That is, either δ does not realise any computable query, or δ realises exactly one computable query, or δ realises a countably infinite set of computable queries. It therefore follows that it is not possible to obtain a finer partition of the class of computable queries without putting restrictions on the set of encodings (for example, by only allowing free encodings). Finally, we have shown that, by adding membership queries to the class of encoding-independent computable queries, the closure of the resulting extended class under composition of mappings is the set of all isomorphism-independent computable queries. Consequently, we can sacrifice order-independence in a computationally complete database language without sacrificing isomorphism-independence. In Figure 8.1 we summarise the inclusion relationship between the classes of computable queries we have investigated, i.e. EI, EIFE, OI, EI+MEM, II, FE, AE and CQ.

It would be interesting to extend our results to allow hereditarily-finite sets as in [GAND80, SAZO93]. We can then extend attribute values to be either atomic values or records, and encodings of records can be correspondingly extended. Let us call such extended records *nested records* and such extended encodings *nested encodings*. Accordingly, an *extended computable query* can now be defined. Informally, an extended computable query is encoding-independent if its computation does not depend on any particular nested encoding used by an ed-oracle. We conjecture that there exists an inclusion relationship amongst the resulting subclasses of extended computable queries, which is analogous to that exhibited in Figure 8.1.

.

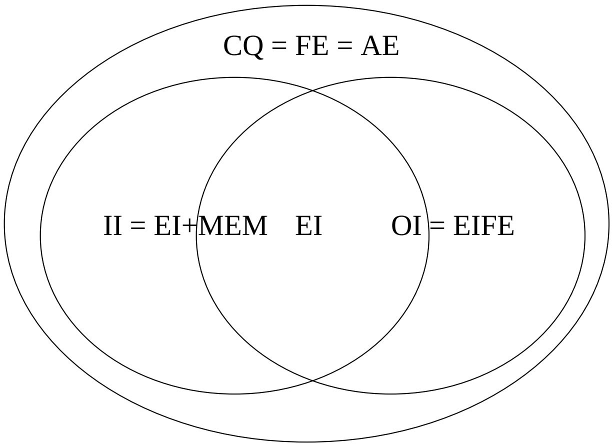

Figure 8.1. The inclusion relationship.

References

[ABIT90] S. Abiteboul and V. Vianu, Procedural languages for database queries and updates, *Journal of Computer and System Sciences* **41** (1990), 181-229.

[ABIT91] S. Abiteboul and V. Vianu, Generic computation and its complexity, *Proceedings of the ACM Symposium on Theory of Computing*, 1991, pp. 209-219.

[AHO79] A.V. Aho and J.D. Ullman, Universality of data retrieval languages, *Proceedings of the ACM Symposium on Principles of Programming Languages* (San-Antonio, Texas), 1979, pp. 110-120.

[CARD91] L. Cardelli and J.C. Mitchell, Operations on records, *Mathematical Structures for Computer Science* **1** (1991), 3-48.

[CHAN80] A.K. Chandra and D. Harel, Computable queries for relational data bases, *Journal of Computer and System Sciences* **21** (1980), 156-178.

[CHAN88] A.K. Chandra, Theory of database queries, *Proceedings of the 7th ACM Symposium on Principles of Database Systems* (Austin, Texas), 1988, pp. 1-9.

[DAVE90] B.A. Davey and H.A. Priestly, *Introduction to Lattices and Order*, Cambridge University Press, Cambridge, U.K., 1990.

[FRIE71] H. Friedman, Algorithmic procedures, generalized Turing machines, and elementary recursion theory, *Proceedings of Logic Colloquium '69* (Manchester, U.K.), 1971, pp. 361-389.

[GAND80] R. Gandy, Church's thesis and the principles for mechanisms, *Proceedings of the Kleene Symposium* (Madison, Wisc.), 1980, pp. 123-148.

[GARE79] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., New York, 1979.

[HALM74] P.R. Halmos, *Naive Set Theory*, Springer-Verlag, New York, 1974.

[HULL93] R. Hull and J. Su, Algebraic and calculus query languages for recursively typed complex objects. *Journal of Computer and System Sciences* **47** (1993), 121-156.

[LEVE95] M. Levene and G. Loizou, A correspondence between variable relations and three-valued propositional logic. *International Journal of Computer Mathematics* **54** (1995), in press.

[RICE53] H.G. Rice, Classes of recursively enumerable sets and their decision problems, *Transactions of the American Mathematical Society* **74** (1953), 358-366.

[ROGE67] H. Rogers Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967.

[SAZO93] V.Y. Sazonov, Hereditarily-finite sets, data bases and polynomial-time computability. *Theoretical Computer Science* **119** (1993), 187-214.

[ULLM88] J.D. Ullman, *Principles of Database and Knowledge-Base Systems, Vol. I,* Computer Science Press, Rockville, Md., 1988.