Henk Csaba

An analysis of the finiteness principle of database theory

Angol nyelvű szakdolgozat (Thesis)

Témavezető (Thesis Adviser): Sain Ildikó

2001
ELTE-TTK

Contents

0 Introduction

As it stems from the very idea, databases have always been supposed to be tractable by computers. There is a very straightforward way of achieving tractability:

- We demand that the relations which may serve as inputs of queries, are finite;
- we consider only those mappings on databases as "computable queries" which are ***finite***, i.e. on a finite input they give a finite answer.

We call this assertion ***the finiteness principle of database theory***.

Starting at the birth of the discipline it was a problem that the very natural candidate for being a query language, the *language of first-order logic* cannot be used for querying in databases in its fully-fledged form, because it violates the finiteness principle.

Thus *if* we accept the finiteness principle, then the best we can do in order to handle this phenomenon is to ask: how could we describe the class of those first-order formulas which do not violate the finiteness principle? (We call such formulas simply *finite formulas.*)

It turns out that we cannot find a direct answer: using a standard argument, we show that the class of finite first-order formulas is not decidable (Theorem 3.7). A more hopeful approach is the one of completeness:

Let us *find a fragment of first-order language which is complete with respect to finiteness* in the following sense: it does not violate the finiteness principle, but it is expressive enough to define all first-order definable finite computable queries.

The answer is at hand: we prove that the very basic query language of *Codd's relational algebra* has this property (Theorem 3.8). Although database theory has developed the background by which this statement can be verified, the emphasis was rather put on another (undecidable) set of first-order formulas: the class of *domain independent formulas.* We also digress on this phenomenon, and we discuss the possible advantages of domain independent formulas over finite ones and vica versa.

After scrutinizing the role of the finiteness principle in classical database theory, we make an attempt to extend the database model beyond finiteness, in order to be able interpret arbitrary first-order queries. Before taking up the enterprise we discuss how such an attempt relates to *practical utilizability.* We have two aims when we set formal criteria (for an extension): on the one hand, to make the usage of the word "extension" logically correct, on the other hand, to fit the basic conditions of practical utilizability. Hence we demand the following criteria to be fulfilled:

***a*)** The basic elements should be the same: some kind of relational structures, on which we execute queries, which are usually defined by query languages. The relation instances in the input must be "of the same kind" as output relations, i.e. queries must be composable.

***b*)** Each considered relation instance must have a finite encoding which enables us decide in linear time whether a given tuple of the universe is included in the instance.

***c*)** We must be able to decide in linear time whether a given relation instance is an old kind of instance, or an extended one. Especially computable queries in the old sense will be computable queries in the extended sense, if we declare them undefined on the extended instances.

***d*)** Any first-order formula must define a total computable query.

***e*)** Complexity of evaluating computable first-order queries must not be affected by this extension.

***f*)** Some computationally complete query language in the old sense must extend naturally to the new instances, yielding a computationally complete query language in the extended sense.

We prove that these criteria are satisfied by the narrowest extension where arbitrary first-order queries are available. We call the relations used in this extension *finitely generated.* During this proof, we show some basic properties of finitely generated relations, which suggest that these may be interesting also from a purely mathematical point of view.

The ***structure of this work*** is the following:

In ***Section 1*** we introduce basic notions, and we establish some basic results like Codd's completeness theorem or domain idependent completeness of relational algebra.

In ***Section 2*** we develop our main tools by investigating the expressive power of relations on a subdomain.

In ***Section 3*** we sketch the concept of computable queries and query languages, also giving a logical characterization of the so-called genericity condition (via being *locally first-order*, Claim 3.6), discuss the relation between domain independent and finite first-order queries, and prove finite completeness of relational algebra.

In the final section, ***Section 4*** we execute the extension of the relational database model to be able to interpret arbitrary first-order queries.

Acknowledgements. I would like to express my gratitude to the following people: for Sain Ildkió, for her patient thesis advisering; for Dr. Benzúr András, for his valuable comments on practical issues; for Szemerey István for his severe efforts on acquiring [AGSS]; and for Hajnal Andréka, Ian Hodkinson, and Janos Makowsky, for supplying me with papers and references, and for answering my questions.

1 A logic-orientated set-up of database concepts

To get started, we present a survey of the basic notions of logic and databases in a way that is suitable to our topic.

Definition 1.1 (Relations, projections, cylindrifications)
Let U, X be sets (the "universe" or "domain", and a "set of attributes" or "variables", respectively). A ***tuple of sort*** X ***over*** U is function $X \longrightarrow U$; the set formed by these will be denoted by U^X. The following example shows the way we refer to tuples: a tuple of sort $\{x, y\}$ can be written in the form $\langle x : a, y : b\rangle$, or if the sort is understood, simply as $\langle a, b\rangle$. If t is a tuple, then we prefer writing t_x for $t(x)$.

A ***relation (instance) of sort*** X ***on*** U is a subset of U^X. (Notice that with this definition a non-empty relation determines its sort: if $R \neq \varnothing$ is a relation of sort X and X' simultaneously, then $X = X'$; whereas $\varnothing$ is a relation of sort X for any set of attributes X. This anomaly will be treated in Definition 1.3, where we will introduce a formal null-value $\varnothing_X$ for relations of sort X over U.) If it does not cause confusion, we do not distinguish relations with a singleton set of attributes from subsets of U.

We say that two functions f, g are ***coherent*** if $f \cup g$ is also a function (considering functions as sets of ordered pairs), i.e. if they coincide on the intersection of intersection domains.

Now we define a quite general version of projection: let R be a relation on U (of sort of some set of attributes X, i.e. $R \subseteq U^X$), and let Y be a set. Then the Y***-projection of*** R (w.r.t. U), $\pi^U_Y R$ is the set of those tuples of sort Y over U which are coherent with some element of R:

$$\pi^U_Y R \stackrel{\mathsf{def}}{=} \{t \in U^Y : (\exists t' \in R)(t,\, t' \text{are coherent})\}.$$

We shall omit the superscript U if it is understood from the context.

Assumig $Y \subseteq X$ we get the ***usual notion of projection***: restricting to the elements of R to a smaller set of attributes. On the other hand, if we assume that $X \subseteq Y$, $\pi_Y R$ gives us the so-called ***dummy representation*** of R (of sort Y): this is the usual way to consider a relation of some "small" arity as one of some bigger arity.

Let us fix a non-empty set Att, intuitively, the collection of the attributes we will use. If $X \subseteq \mathsf{Att}$, then we define the unary operation $\mathsf{c}^{\perp U}_{(X)}\, R$ on the U-relations of sort Att, called the (X)***-ortho-cylindrification*** (w.r.t. U), as follows. For $R \subseteq U^{\mathsf{Att}}$ we let $\mathsf{c}^{\perp U}_{(X)}\, R = \pi_{\mathsf{Att}} \pi_X R$ (the dummy representation of the X-projection of R). We shall omit the superscript U from $\mathsf{c}^{\perp U}_{(X)}$ if it is understood from the context. We also define ***the operation*** $\mathsf{c}_{(X)}$ ***of*** (X)***-cylindrification*** as $\mathsf{c}_{(X)} = \mathsf{c}^{\perp}_{(\mathsf{Att} \smallsetminus X)}$. Furthermore, we will use the following abbreviations: for $x \in \mathsf{Att}$, $\mathsf{c}^{\perp}_x = \mathsf{c}^{\perp}_{(\{x\})}$ and $\mathsf{c}_x = \mathsf{c}_{(\{x\})}$ (while, if Att is the non-negative integer n, and $i < n$ is also a non-negative integer, then, according to von Neumann's concept of ordinals, $\mathsf{c}_{(i)} = \mathsf{c}_{(\{0,\ldots,i-1\})}$). We say that R is (X)***-closed*** (x***-closed***), if $R = \mathsf{c}_{(X)}\, R$ ($R = \mathsf{c}_x\, R$). A synonym for "R is x-closed" is "R ***is dummy in attribute*** x" (i.e. R carries no information about attribute x). The ***dimension set*** of R is $\Delta R = \{x \in \mathsf{Att} : R \text{ is not dummy in } x\}$.

We note that c_x in some sense (to be explained later) is the algebraic counterpart of existential quantification ("on the variable x"), as it is suggested by the following direct definition:

$$\mathsf{c}_x\, R = \left\{t \in U^{\mathsf{Att}} : (\exists u \in U)(\text{modifying the value of } t \text{ at } x \text{ to } u \text{ we get to } R)\right\}. \qquad \square$$

Definition 1.2 (Cylindric algebras)
For a given set Att of attributes, we define the ***cylindric similarity type of dimension*** Att, $\mathsf{CA}_{\mathsf{Att}}$-type for short, to consist of the Boolean operation symbols $(0, 1, \cap, \cup, \smallsetminus)$, the unary cylindrification symbols (c_x, where $x \in \mathsf{Att}$), and the identity constant symbols (d_{xy}, where $x, y \in \mathsf{Att}$). (We use the following convention: the Boolean symbol $\smallsetminus$ stands for the unary complementation operator, but we use the abbreviation $r \smallsetminus q = r \cap (\smallsetminus q)$). In the literature of cylindric algebras Att is usually chosen to be some ordinal $\leqslant \omega$.

The ***full cylindric set algebra*** of dimension Att over U, denoted by $\mathfrak{P}_{\mathsf{CA}}\left(U^{\mathsf{Att}}\right)$, is the algebra of type $\mathsf{CA}_{\mathsf{Att}}$ with base set $\mathcal{P}\left(U^{\mathsf{Att}}\right)$, where the Boolean operation symbols are interpreted to be the usual set operations, c_x is interpreted by c_x^U, and d_{xy} is interpreted by $\{t \in U^{\mathsf{Att}} : t_x = t_y\}$.

A ***cylindric set algebra*** (of dimension Att), a $\mathsf{Cs}_{\mathsf{Att}}$ for short, is a subalgebra of $\mathfrak{P}_{\mathsf{CA}}\left(U^{\mathsf{Att}}\right)$ for some set Att. We use the notation $\mathsf{Cs}_{\mathsf{Att}}$ also to refer to the class of all cylindric set algebras of dimension Att. The variety generated by $\mathsf{Cs}_{\mathsf{Att}}$ is called the variety of ***representable cylindric algebras*** (of dimension Att), denoted by $\mathsf{RCA}_{\mathsf{Att}}$.

Next we define some more derived operations:

The way universal quantification can be obtained from existential quantification ($\forall x\, \varphi \iff \neg\exists x\, \neg\varphi$) suggests us how to define the algebraic counterpart of universal quantification ("on the variable x"): it is called the ***x-th axification*** in this work, and is given by the term $\mathsf{a}_x\, r \overset{\mathsf{def}}{=} \smallsetminus \mathsf{c}_x\, (\smallsetminus r)$ (in [HMT1] these are called inner cylindrifications and are denoted by c_i^{∂}).

We define ***substitutions*** by the term $\mathsf{s}_y^x\, r = \mathsf{c}_x(r \cap \mathsf{d}_{xy})$ (where $x, y \in \mathsf{Att}$. We give an example (with a somewhat clumsy notation) to show the intuition behind this notion: if $R(x, y)$ is (the dummy representation of) a relation on U of sort $\{x, y\}$, then in $\mathfrak{P}_{\mathsf{CA}}\left(U^{\mathsf{Att}}\right)$ $\mathsf{s}_z^y\, R(x, y) = R(x, z)$, $\mathsf{s}_x^y\, R(x, y) = R(x, x)$ — if we consider $R(x, y)$ as a directed graph on U, this latter is the set of those points which have a loop.

It is straightforward to generalize other relational notions (like dimension set) to CA's. Thus we may use these notions to CA's without further discussion. □

Definition 1.3 (Relational structures, formulas, meanings, equivalence)
In this definition we present some basic notions of first-order model theory, custom-tailored to database theory and our puroses.

Royughly speaking, by a ***relational type*** (with attributes from Att) we mean a list of relation symbols (or schemes) with a sort assignment. Formally it is a function the values of which are finite subsets of Att. The elements of the domain of a relational type are called the *relation symbols* of τ. Slightly abusing the notation, we express "R is a realtion symbol of τ" by writing $R \in \tau$. We emphasize that we do not exclude the empty sort. Now we fix such a τ.

We use the usual set of first-order (logical) connectives: $\mathbf{T}$ ("true"), $\mathbf{F}$ ("false") are 0-ary, $\wedge$, $\vee$ are binary, $\neg$, $\exists x$, $\forall x$ $\quad (x \in \mathsf{Att})$ are unary, $x = y$ $\quad (x, y \in \mathsf{Att})$ are 0-ary connectives.

We assume that the reader is familiar with the basic notions of first-order syntax and semantics. Here we present the definitions of this notions, however, not in the usual, step-by-step recursive way, but in a way that emphasizes their algebraic features.

By the *formula algebra* of type τ, denoted by Fml_τ, we mean the word algebra (absolutely free algebra) with the connectives as operations, and with the relation symbols of τ as generators. The elements of Fml_τ are called ***formulas (of type τ)***.

If τ is understood from the context, we simply write Fml. In order to distinguish the equality of formulas from the connective "$x = y$", the equality of formulas will be denoted by the symbol "$\approx$"; thus we may give declarations like "let $\varphi \approx x = y$".

A ***relational structure*** of type τ is a pair $\mathfrak{M} = \langle U, I\rangle$, where U is a non-empty set, and I (the so-called *interpretation*) is a function which assigns a relation of sort $\tau(\mathsf{R})$ on U to each $\mathsf{R} \in \tau$. We will use the following convention: as any time up to this point, we will denote relations by normal capitals like $R, Q, \ldots$, and relation symbols by serif letters like $\mathsf{R}, \mathsf{Q}, \ldots$ If R is present, it is understood that R refers to an iterpretation of R. Now we fix such an $\mathfrak{M}$.

Next we give some variants of semantics, by means of various *meaning functions*, which map the formulas to "somewhere", which "somewhere" will be related to $\mathfrak{M}$ in some natural way. We will also define the set of free variables of a formula analogously.

Here we introduce a convention. If $\mathfrak{A} = \langle A, f_1, \ldots, f_n\rangle$ and $\mathfrak{B} = \langle B, g_1, \ldots, g_n\rangle$ are two algebraic structures, where f_i is of the same arity as g_i, then by a $\left(\begin{smallmatrix} A, & f_1, & \ldots, & f_n \\ B, & g_1, & \ldots, & g_n\end{smallmatrix}\right)$ *-homomorphism*, or $\left(\begin{smallmatrix} f_1, & \ldots, & f_n \\ g_1, & \ldots, & g_n\end{smallmatrix}\right)$ *-homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$* we mean a function $q : A \longrightarrow B$ which associates g_i to f_i (that is, if f_i, g_i are k-ary, then $q(f_i(x_1, \ldots, x_k)) = g_i(q(x_1), \ldots, q(x_k))$ is satisfied).

- $\mathsf{mng}_{\mathfrak{M}} : \mathsf{Fml} \longrightarrow \mathfrak{P}_{\mathsf{CA}}(U^{\mathsf{Att}})$, the ***(first-order) meaning function of*** $\mathfrak{M}$, is the uniquely determined

$$\begin{pmatrix} \mathbf{T}, & \mathbf{F}, & \wedge, & \vee, & \neg, & \exists x, & \forall x, & x = y \\ 1, & 0, & \cap, & \cup, & \smallsetminus, & \mathsf{c}_x, & \mathsf{a}_x, & \mathsf{d}_{xy} \end{pmatrix}_{x,y \in \mathsf{Att}}$$

  -homomorphism, which takes each $\mathsf{R} \in \tau$ to $\pi_{\mathsf{Att}} I(\mathsf{R})$. The image of $\mathsf{mng}_{\mathfrak{M}}$ is called the ***meaning algebra of*** $\mathfrak{M}$, and is denoted by $\mathsf{mng}(\mathfrak{M})$.

- Here id will denote the identity function (of a set understood from the context). frv, called "the mapping which associates to each $\varphi \in \mathsf{Fml}$ the set of its free variables", or ***free variable-mapping*** for short, is the uniquely determined

$$\begin{pmatrix} \mathsf{Fml}, & \mathbf{T}, & \mathbf{F}, & \wedge, & \vee, & \neg, & \exists x, & \forall x, & x = y \\ \mathcal{P}(\mathsf{Att}), & \varnothing, & \varnothing, & \cup, & \cup, & \mathsf{id}, & X \mapsto X \smallsetminus \{x\}, & X \mapsto X \smallsetminus \{x\}, & \{x, y\} \end{pmatrix}_{x,y \in \mathsf{Att}}$$

  -homomorphism, which takes $\mathsf{R} \in \tau$ to $\tau(\mathsf{R})$. "$x \in \mathsf{frv}(\varphi)$" is to be read as "$x$ is free in φ"; we will use the notation "$\varphi(x_1, \ldots, x_k)$" to express that "$\varphi$ is a formula and $\mathsf{frv}(\varphi) = \{x_1, \ldots, x_k\}$".

- First for each sort $X \subseteq \mathsf{Att}$ we introduce a new symbol $\varnothing_X$, called ***the empty relation of sort*** X. Now the ***database meaning function of*** $\mathfrak{M}$, $\mathsf{d\text{-}mng}_{\mathfrak{M}}$ is defined as follows: for $\varphi \in \mathsf{Fml}$,

$$\mathsf{d\text{-}mng}_{\mathfrak{M}}(\varphi) \stackrel{\mathsf{def}}{=} \begin{cases} \pi_{\mathsf{frv}(\varphi)} \mathsf{mng}_{\mathfrak{M}}(\varphi) & \text{if } \mathsf{mng}_{\mathfrak{M}}(\varphi) \neq \varnothing \\ \varnothing_{\mathsf{frv}(\varphi)} & \text{if } \mathsf{mng}_{\mathfrak{M}}(\varphi) = \varnothing \end{cases}.$$

  (An alternative, logically more succint, essentially equivalent formulation of database meaning is the following:

$$\mathsf{d\text{-}mng}'_{\mathfrak{M}} \stackrel{\mathsf{def}}{=} \langle \mathsf{mng}_{\mathfrak{M}}, \mathsf{frv}\rangle : \mathsf{Fml} \longrightarrow \mathfrak{P}_{\mathsf{CA}}(U^{\mathsf{Att}}) \times \mathcal{P}(\mathsf{Att}); \qquad \square$$

however, because of practical reasons, we chose the above formalization of database meaning — e.g. if $\mathsf{R} \in \tau$ is a relation symbol with non-empty interpretation, then $\mathsf{d\text{-}mng}_{\mathfrak{M}}(\mathsf{R})$ is simply $I(\mathsf{R})$, meanwhile $\mathsf{d\text{-}mng}'_{\mathfrak{M}}(\mathsf{R}) = \langle \pi_{\mathsf{Att}} I(\mathsf{R}), \tau(\mathsf{R})\rangle$, which is a more complex object.)

- The ***reduced database meaning*** of a formula φ in $\mathfrak{M}$ is $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi) \stackrel{\mathsf{def}}{=} \pi_{\mathsf{frv}(\varphi)}\, \mathsf{mng}_{\mathfrak{M}}(\varphi)$. We say that a relation on U is ***(first-order) expressible***, if it is of the form $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi)$ for some formula φ.

Without formal definition we will use the usual notation for variable substitution, e.g. if $\tau(\mathsf{R}) = \{x, y\}$, we may write $\mathsf{R}(x, z)$ and $\mathsf{R}(x, x)$. (These operations are expressible by the connectives $\exists x$, $x = y$, as (in the algebraic context) s^x_y shows it, see at the end of Definition 1.2.)

In the sequel we still keep τ fixed, but not $\mathfrak{M}$.

Motivated by database theory, we introduce a notion of ***formula equivalence*** slightly deviating from the usual one:

Let $\mathcal{K}$ be a class of relational structures (of type τ). For $\varphi, \psi \in \mathsf{Fml}$ we say that φ and ψ are t(yped) equivalent in $\mathcal{K}$ if for each $\mathfrak{M} \in \mathcal{K}$, $\mathsf{d\text{-}mng}_{\mathfrak{M}}(\varphi) = \mathsf{d\text{-}mng}_{\mathfrak{M}}(\psi)$. If $\mathcal{K}$ is chosen to be the class of all relational structures, we can simply say that φ and ψ are (logically, typed) equivalent.

The usual notion of formula equivalence – in this work being called ***non-typed equivalence*** – is as follows: φ and ψ are non-typed equivalent (in $\mathcal{K}$) if for each $\mathfrak{M}$ ($\in \mathcal{K}$) $\mathsf{mng}_{\mathfrak{M}}(\varphi) = \mathsf{mng}_{\mathfrak{M}}(\psi)$.

φ and ψ are equivalent in some $\mathcal{K}$ iff they are non-typed equivalent and $\mathsf{frv}(\varphi) = \mathsf{frv}(\psi)$.

Remark 1.4 The reader may see that in Definition 1.3 we did not mention an important feature of logical systems: *validity*. This corresponds to putting the emphasis on queries (instead of dependencies), as it often happens in database theory, and especially, in this work. The connection between meaning and validity (in the usual first-order sense) is the following: for $t \in U^{\mathsf{Att}}$, $\mathfrak{M} \models \varphi[t] \iff t \in \mathsf{mng}_{\mathfrak{M}}(\varphi)$ (thus "validity in $\mathfrak{M}$ under an evaluation t" stands closer to the meaning-notion of general logics than to validity) and $\mathfrak{M} \models \varphi \iff \mathsf{mng}_{\mathfrak{M}}(\varphi) = U^{\mathsf{Att}}$.

With the above definition of $\mathsf{d\text{-}mng}_{\mathfrak{M}}$, $\varnothing_X$ serves as a null-value for the relations of sort X — thus for any $\varphi \in \mathsf{Fml}$, $\mathsf{d\text{-}mng}_{\mathfrak{M}}(\varphi)$ determines $\mathsf{frv}(\varphi)$, so saying that "$\mathsf{d\text{-}mng}$ and $\mathsf{d\text{-}mng}'$ are different formulations of the same thing" is correct. (The connection between $\mathsf{mng}_{\mathfrak{M}}$ and frv is the following: for $\varphi \in \mathsf{Fml}$, $\Delta\, \mathsf{mng}_{\mathfrak{M}}(\varphi) \subseteq \mathsf{frv}(\varphi)$; we cannot make stronger restrictions, e.g. $\varnothing$ or U^{Att} can be the meaning of any relation symbol.) However, in the current investigations the type ambiguity of the empty relation would not cause trouble, since we can always refer directly to $\mathsf{frv}(\varphi)$.

There is a (for our purpose) purely aesthetical reason for making null-values for every sort: this way, as one expects from a reasonable meaning-function, $\mathsf{d\text{-}mng}$ is *compositional*, i.e. it is a homomorphism (if we declare the operations in its range as it stands for reason), or saying it more precisely, $\mathsf{Ker}\,(\mathsf{d\text{-}mng}_{\mathfrak{M}})$ is a congruence of the formula algebra. For example, if φ, ψ are formulas with the same database meaning, then the database meaning of $\neg\varphi$ and $\neg\psi$ also coincide. On the other hand, the reduced database meaning is not compositional: it may happen that for some $R(x), Q(x, y) \in \tau$ $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(R(x)) = I(R(x)) = \varnothing$ and also $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(Q(x, y)) = I(Q(x, y)) = \varnothing$, whilst $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\neg R(x)) = U^{\{x\}} \neq U^{\{x,y\}} = \mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\neg Q(x, y))$.

Thus we could formulate and algebraize some kind of "database logic", where the algebraic counterpart would be "the algebra of relations with attribute set $\subseteq$ Att", i.e. declaring $\mathcal{P}^*\left(U^X\right) \stackrel{\mathsf{def}}{=} \mathcal{P}\left(U^X\right) \setminus \{\varnothing\} \cup \{\varnothing_X\}$, $\mathcal{P}^*\left(U^{\subseteq \mathsf{Att}}\right) \stackrel{\mathsf{def}}{=} \bigcup_{X \subseteq \mathsf{Att}} \mathcal{P}^*\left(U^X\right)$ can be made to a $\mathsf{CA}_{\mathsf{Att}}$-type algebra in the natural way. However, this kind of algebra (and logic) lacks some usual good property of CA's (and usual first order logic): e.g. here $\cap, \cup$ are not lattice operations, since, for example, if $\mathsf{R}(x), \mathsf{Q}(y) \in \tau$, then $\mathsf{d\text{-}mng}_{\mathfrak{M}}(\mathsf{R}(x)) = \{x : R(x)\}$, whereas $\mathsf{d\text{-}mng}_{\mathfrak{M}}\left((\mathsf{R}(x) \vee \mathsf{Q}(y)) \wedge \mathsf{R}(x)\right) = \{\langle x, y\rangle : R(x)\}$.

Our notion of reduced database meaning follows that variant of first-order logic which is considered in [AHV].

The reader seeking for more information on general logical systems, meanings, and algebraization of logics is referred to [ANS a]. □

Definition 1.5 (Databases, queries, active domain)
Again we fix a relational type τ.

We say that a relational structure $\mathfrak{D} = \langle U, I\rangle$ is a ***database***, if U is infinite, and for each $\mathsf{R} \in \tau$, $I(\mathsf{R})$ is finite.

Traditional database terminology uses the notation $\{\langle x_1, \ldots, x_n\rangle : \varphi(x_1, \ldots, x_n)\}$ for denoting queries.

This notation and this word is used there in the following senses:

- this may refer to the mapping from database instances to relation instances, given by the semantics of φ;

- this may refer to the syntactical entity, which is not, but is to be interpreted over database instances; which we use as a "name" for a mapping as above.

Accordingly, we will use the expression "***query***" in the following senses (we will not use the above notation):

- By "*the query defined (or realized) by the formula* φ" we mean the mapping $\mathfrak{D} \mapsto \mathsf{d\text{-}mng}^-_{\mathfrak{D}}(\varphi)$ (where $\mathfrak{D}$ is a database). This mapping will be denoted by $\mathsf{d\text{-}mng}^-(\varphi)$. A ***first-order query*** is a mapping of the form $\mathsf{d\text{-}mng}^-(\varphi)$. By "the *generalized* query defined by φ" we mean the mapping $\mathfrak{M} \mapsto \mathsf{d\text{-}mng}_{\mathfrak{M}}(\varphi)$ (where $\mathfrak{M}$ is an arbitrary relational structure).

- If we say "*φ is a query (of type τ)*" then we mean that $\varphi \in \mathsf{Fml}$, besides, it is implicitely understood that we consider only the database instances of φ. E.g. if we say "φ, ψ are equivalent queries", or „the formulas φ, ψ are equivalent as queries" we mean that φ and ψ are formulas, and they are equivalent in the class of databases. Thus $\exists x\, \neg \mathsf{R}(x)$ and $\mathbf{T}$ are equivalent queries, but are not equivalent formulas.

A ***query*** φ ***is finite*** if in every database $\mathfrak{D}$, $\mathsf{d\text{-}mng}_{\mathfrak{D}}(\varphi)$ is finite. The importance of this notion is the following: *the query defined by a first order formula φ is a (computable) query in the sense of* [AHV] *iff φ is a finite query.* Soon we will give a simple characterization of finite queries. To be able to do so, we introduce the notion of an *active domain*. Intuitively, the active domain of a formula φ (in a given relational structure) is the set of those elements of the universe which occur in φ, or we can say, it is the narrowest set over which φ makes sense without any modifications, without establishing conventions.

If $\mathfrak{M}$ is a relational structure and R is a relation symbol, then the ***active domain*** of R in $\mathfrak{M}$, denoted by $\mathsf{adom}_{\mathfrak{M}}(\mathsf{R})$, is the set of those elements of the universe which occur in R, formally $\mathsf{adom}_{\mathfrak{M}}(\mathsf{R}) = \bigcup_{x\in\tau(\mathsf{R})} \pi_{\{x\}} I(\mathsf{R})$ (here we use an $U^{\{x\}} = U$ identification). Besides, $\mathsf{adom}_{\mathfrak{M}}(\mathbf{T}) = \mathsf{adom}_{\mathfrak{M}}(\mathbf{F}) = \mathsf{adom}_{\mathfrak{M}}(x = y) = \varnothing$, and from this $\mathsf{adom}_{\mathfrak{M}}$ is defined recursively as $\mathsf{adom}_{\mathfrak{M}}(\neg\varphi) = \mathsf{adom}_{\mathfrak{M}}(\exists x\, \varphi) = \mathsf{adom}_{\mathfrak{M}}(\forall x\, \varphi) = \mathsf{adom}_{\mathfrak{M}}(\varphi)$, and $\mathsf{adom}_{\mathfrak{M}}(\varphi \wedge \psi) = \mathsf{adom}_{\mathfrak{M}}(\varphi \vee \psi) = \mathsf{adom}_{\mathfrak{M}}(\varphi) \cup \mathsf{adom}_{\mathfrak{M}}(\psi)$. We can also speak of the active domain of a *relational structure* $\mathfrak{M}$: $\mathsf{adom}(\mathfrak{M}) = \bigcup_{\mathsf{R}\in\tau} \mathsf{adom}_{\mathfrak{M}}(\mathsf{R})$. □

Remark 1.6 Our notion of a database follows the one of [AHV], except that we omitted those conditions which are irrelevant for our purpose: e.g., we did not demand that the type consists of only finitely many relation symbols, we did not demand the presence of constants, etc.

A first-order query is not necessarily a computable query in the sense of Definition 3.1: it is just a quasi-query. □

We give a simple characterization if finite queries:

Definition 1.7 A formula φ has the adom***-preserving property*** in the class $\mathcal{K}$ of relational structures, if for each relational structure $\mathfrak{M} \in \mathcal{K}$, $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi) \subseteq (\mathsf{adom}_{\mathfrak{M}}(\varphi))^{\mathsf{frv}(\varphi)}$. □

Claim 1.8 *A query φ is finite iff it has the* adom*-preserving property in databases.*

Proof. Since the active domains of queries are finite, it is straightforward that the adom-preserving property in databases implies finiteness.

To see the other direction, first we state the *automorphism invariance of meanings*:

If α is a function on U and $R \subseteq U^X$, then by $\alpha(R)$ we mean $\{\alpha \circ t : t \in R\}$. Now, if α is an automorphism of a relational structure $\mathfrak{M}$, then for each $\varphi \in \mathsf{Fml}$ $\alpha(\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi)) = \mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi)$. (This is also valid for all meaning-notions concerned so far; verifiying this is left to the reader.)

Now assume φ is a finite query, but we have a database $\mathfrak{D} = \langle U, I\rangle$ such that $\mathsf{d\text{-}mng}^{-}_{\mathfrak{D}}(\varphi) \not\subseteq (\mathsf{adom}_{\mathfrak{D}}(\varphi))^{\mathsf{frv}(\varphi)}$; we may assume that those relation symbols which do not occur in φ have empty interpretations. Let u be an element of U which occurs in $\mathsf{d\text{-}mng}^{-}_{\mathfrak{D}}(\varphi)$ but is out of $\mathsf{adom}_{\mathfrak{D}}(\varphi)$. Since U is infinite and $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi)$ is finite, we can find a $u' \in U$ which does not occur in $\mathsf{adom}_{\mathfrak{D}}(\varphi)$. Let α be that transposition on U which interchanges u and u'. Then α is an automorphism of $\mathfrak{D}$, but $\alpha(\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi)) \neq \mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi)$, since u' occurs only in the left one. This contradicts automorphism invariance. ■

Remark 1.9 The adom-preserving property in databases (thus finiteness) does not imply the general adom-preserving property (i.e. in all relational structures): e.g. $\varphi \approx \neg\mathsf{R}(x) \wedge \exists! x\, \neg\mathsf{R}(x)$ has empty $\mathsf{d\text{-}mng}^{-}_{\mathfrak{D}}$ for each database $\mathfrak{D}$, but in $\mathfrak{M} = \langle\{u\}, \mathsf{R} \mapsto \varnothing\rangle$ $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi) = \{u\}$, whereas $\mathsf{adom}_{\mathfrak{M}}(\varphi) = \varnothing$. □

Definition 1.10 (Codd formulas)
We define the following derived (partial) connectives called ***Codd connectives***:

- $\mathbf{T}$, $\mathbf{F}$, $\wedge$ are Codd connectives.
- For $x \in \mathsf{Att}$, $\exists x$ restricted to those φ's, where $x \in \mathsf{frv}(\varphi)$, is a Codd connective.

- $\vee$ restricted to those pairs φ, ψ, where $\mathsf{frv}(\varphi) = \mathsf{frv}(\psi)$, is a Codd connective.

- $(\wedge\neg)$ is a binary Codd connective. It can be applied to those pairs φ, ψ, where $\mathsf{frv}(\varphi) = \mathsf{frv}(\psi)$, and $(\wedge\neg)(\varphi, \psi) \overset{\mathsf{def}}{\approx} \varphi \wedge \neg\psi$.

- For $x, y \in \mathsf{Att}$, $(x = y \wedge\ )$ is a unary Codd connective. It can be applied to φ if $\mathsf{frv}(\varphi) \cap \{x, y\} \neq \varnothing$, and $(x = y \wedge\ )(\varphi) \overset{\mathsf{def}}{\approx} x = y \wedge \varphi$.

A formula φ is called a ***Codd formula***, if it is produced from the relation symbols by the Codd connectives. The set of Codd formulas is denoted by $\mathsf{CFml} = \mathsf{CFml}_\tau$. A ***Codd query*** is a first-order query of the form $\mathsf{d\text{-}mng}^-(\varphi)$, where φ is a Codd formula.

□

Now we will establish some basic facts concerning Codd formulas.

We can state immediately the following observation:

Claim 1.11 *All Codd queries are finite.*

Proof. The proof is left to the reader. ■

One of our major purposes is to see the other direction, that is, each finite query is equivalent to a Codd query (see Theorem 3.8).

There are some formulas which are not literally Codd formulas but still can be treated as *abbreviations* of Codd formulas:

Remark 1.12 (Extending Codd formulas, "the empty structure")

- If $\varphi, \psi \in \mathsf{CFml}$ such that $\mathsf{frv}(\varphi) \supseteq \mathsf{frv}(\psi)$, then $\varphi \wedge \neg\psi$ is equivalent to the Codd formula $\varphi \wedge \neg(\varphi \wedge \psi)$; if $\varphi \in \mathsf{CFml}$ such that $\mathsf{frv}(\varphi) = \varnothing$ then $\neg\varphi$ is equivalent to the Codd formula $\mathbf{T} \wedge \neg\varphi$.

- If $\varphi(x_1, \ldots, x_n) \in \mathsf{CFml}$, then $\varphi(y, x_2, \ldots, x_n)$ is equivalent to the Codd formula $\exists x_1(y = x_1 \wedge \varphi(x_1, \ldots, x_n))$.

- If $\mathfrak{M} = \langle U, R_i \rangle_i$, and $D \subseteq U$, let $\mathfrak{M}\restriction_D = \mathfrak{M} = \left\langle U, R_i \cap D^{\tau(\mathsf{R}_i)} \right\rangle_i$. If $\delta(x) \in \mathsf{CFml}$ and $\varphi \in \mathsf{Fml}$, then we will define a Codd formula $\varphi\restriction_\delta$, the ***relativization of φ w.r.t.*** δ, having the following property: for each relation structure $\mathfrak{M}$ where $\mathsf{d\text{-}mng}_{\mathfrak{M}}(\delta)$ is not empty, $\mathsf{d\text{-}mng}_{\mathfrak{M}}\left(\varphi\restriction_\delta\right) = \mathsf{d\text{-}mng}_{\mathfrak{M}_\delta}(\varphi)$, where $\mathfrak{M}_\delta = \mathfrak{M}\restriction_{\mathsf{d\text{-}mng}_{\mathfrak{M}}(\delta)}$; $\varphi\restriction_\delta$ is given by the following recursive definition:

  - $\mathbf{T}\restriction_\delta \approx \mathbf{T}$, $\mathbf{F}\restriction_\delta \approx \mathbf{F}$, $(x = y)\restriction_\delta \approx x = y \wedge \delta(x) \wedge \delta(y)$ $(x, y \in \mathsf{Att})$, and for $\mathsf{R}(x_1, \ldots, x_n) \in \tau$, $\mathsf{R}\restriction_\delta \approx \mathsf{R} \wedge \delta(x_1) \wedge \ldots \wedge \delta(x_n)$;
  - for $\varphi(x_1, \ldots, x_n)$, $\psi(y_1, \ldots, y_m) \in \mathsf{Fml}$, $x \in \mathsf{Att}$, $(\varphi \wedge \psi)\restriction_\delta \approx \varphi\restriction_\delta \wedge \psi\restriction_\delta$, $(\varphi \vee \psi)\restriction_\delta \approx \left(\varphi\restriction_\delta \wedge \delta(y_1) \wedge \ldots \wedge \delta(y_m)\right) \vee \left(\psi\restriction_\delta \wedge \delta(x_1) \wedge \ldots \wedge \delta(x_n)\right)$, $(\neg\varphi)\restriction_\delta \approx (\delta(x_1) \wedge \ldots \wedge \delta(x_n)) \wedge \neg\varphi\restriction_\delta$, $(\exists x\, \varphi)\restriction_\delta \approx \exists x \left(\varphi\restriction_\delta \wedge \delta(x)\right)$, and $(\forall\, x\varphi)\restriction_\delta \approx (\neg\exists x\, \neg\varphi)\restriction_\delta$.

  (Checking that $\varphi\restriction_\delta$ has the above mentioned property is left to the reader.) Without defining the notion of "empty structure", we define formally "the database meaning of φ in the empty structure", denoted by $\mathsf{d\text{-}mng}_\circ(\varphi)$ as follows: it is $\mathsf{d\text{-}mng}_{\mathfrak{M}}\left(\varphi\restriction_\delta\right)$, where $\mathfrak{M}$ is an arbitrary relational structure, and $\delta(x)$ is an arbitrary formula such that $\mathsf{mng}_{\mathfrak{M}}(\delta) = \varnothing$ (this is well defined)*. ($\mathsf{d\text{-}mng}^{-}_{\mathsf{o}}$ is meant similarly.) In case of such $\mathfrak{M}$, δ using $\mathfrak{M}_\delta \stackrel{\mathsf{def}}{=} \mathsf{o}$ (this is a *syntactical* entity), now we have that *for any* relational structure $\mathfrak{M}$ $\mathsf{d\text{-}mng}_{\mathfrak{M}}\left(\varphi\lceil_\delta\right) = \mathsf{d\text{-}mng}_{\mathfrak{M}_\delta}(\varphi)$.

- for $\varphi \in \mathsf{Fml}$, using the definition of $\mathsf{adom}_{\mathfrak{M}}$, it is straightforward to develop a Codd formula $\mathsf{adom}_\varphi(x)$ such that for each relational structure $\mathfrak{M}$ $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\mathsf{adom}_\varphi) = \mathsf{adom}_{\mathfrak{M}}(\varphi)$. Provided $\mathsf{adom}(\mathfrak{M})$ is finite, or the type τ is finite, we can develop a Codd-formula, also denoted by $\mathsf{adom}(\mathfrak{M})$ such that its reduced database meaning in $\mathfrak{M}$ is the set $\mathsf{adom}(\mathfrak{M})$. The details of these constructions are left to the reader.

Thus the above listed formulas also will be treated as Codd formulas. □

As now we have the "semantics of the empty structure", we can set the following definition:

Definition 1.13 We say a formula φ is ***domain independent*** in a class of relational structures $\mathcal{K}$, if for each $\mathfrak{M} = \langle U, I\rangle$, $\mathfrak{M} \in \mathcal{K}$, and $D \subseteq U$ such that $\mathsf{adom}_{\mathfrak{M}}(\varphi) \subseteq D$ we have $\mathsf{d\text{-}mng}_{\mathfrak{M}\lceil_D}(\varphi) = \mathsf{d\text{-}mng}_{\mathfrak{M}}(\varphi)$. And we say simply that φ is domain independent, if it is domain independent in the class of all relational structures. □

Of course, domain independent queries are finite, because we get the same result when evaluating them on their active domain, which is finite in databases; the converse does not hold, since the query $\exists x\, \neg \mathsf{R}(x)$ is finite but not domain independent.

The following claim is a classical result of database theory, see e.g. [AHV]:

Claim 1.14 *If a formula φ is domain independent in $\mathcal{K}$, then there is a Codd formula $\widetilde{\varphi}$ which is equivalent with φ in $\mathcal{K}$ — namely $\varphi\lceil_{\mathsf{adom}_\varphi}$. (Note that Codd formulas are domain independent.)* ■

We will use the notation $\mathcal{P}\left(U^{\subseteq \mathsf{Att}}\right) \stackrel{\mathsf{def}}{=} \bigcup_{X\subseteq \mathsf{Att}} \mathcal{P}\left(U^X\right)$.

Remark 1.15 (Codd algebra) Codd formulas are a kind of logization of Codd's relational algebra. This connection is discussed below; here we recall what Codd algebra is, based on the definition of [AHV]. Codd algebra was introduced in [Codd70] and [Codd72].

Codd algebra consists of the following partial operations on $\mathcal{P}\left(U^{\subseteq \mathsf{Att}}\right)$ or $\mathcal{P}^*\left(U^{\subseteq \mathsf{Att}}\right)$ (in the latter case $\varnothing_X$, the null-value for $\mathcal{P}^*\left(U^X\right)$ is to be treated as it stands for reason):

Let R be a relation of sort X and Q be a relation of sort Y over U (for some $X, Y \subseteq \mathsf{Att}$). Then

- for $Z \subseteq \mathsf{Att}$ the *projection* π_Z can be applied to R if $Z \subseteq X$;

*Consider that if $X \neq \varnothing$, then only $\varnothing$ is a relation of sort X on $\varnothing$, whereas there are two relations of sort $\varnothing$ on $\varnothing$: $\varnothing$ and $\{\langle\rangle\}$ (where $\langle\rangle$ is the *empty tuple*, formally the same as $\varnothing$). Hence if we tried to define the empty structure o as such, we would get that for any $\varphi \in \mathsf{Fml}$ $\mathsf{mng}_{\mathsf{o}}(\varphi) = \varnothing$ (even for $\varphi \approx \mathbf{T}$); this means $\mathsf{d\text{-}mng}_{\mathsf{o}}$, as we defined here (by which $\mathsf{d\text{-}mng}_{\mathsf{o}}(\mathbf{T}) = \{\langle\rangle\}$), and the $\mathsf{d\text{-}mng}$-notion of Definition 1.3, derived form mng and frv, would give different result in case of the empty structure. However, using a direct defintion for $\mathsf{d\text{-}mng}$ (saying, it is the e.g. $\mathsf{d\text{-}mng}_{\mathfrak{M}}(\varphi \wedge \psi) = \pi_{\mathsf{frv}(\varphi)\cup\mathsf{frv}(\psi)}\varphi \cap \pi_{\mathsf{frv}(\varphi)\cup\mathsf{frv}(\psi)}\psi$), this definition would produce our $\mathsf{d\text{-}mng}_{\mathsf{o}}$. This is the motivation of our way to introduce $\mathsf{d\text{-}mng}_{\mathsf{o}}$: correct this slight anomaly of the $\mathsf{d\text{-}mng}$-notion derived form mng, frv.

- the *join* of R, Q is defined as $R \bowtie Q = \pi_{X\cup Y}(\pi_{\mathsf{Att}} R \cap \pi_{\mathsf{Att}} Q) = \pi_{X\cup Y} R \cap \pi_{X\cup Y} Q$;
- for $x, y \in \mathsf{Att}$, the $(x = y)$-*selection* can be applied to R if $x, y \in X$ and is defined as $\sigma_{x=y} R = R \bowtie \{\langle x : a, y : a\rangle : a \in U\} = \pi_X(\pi_{\mathsf{Att}} R \cap \mathsf{d}_{xy})$;
- for $x, y \in \mathsf{Att}$, the $(x \to y)$-*renaming* can be applied to R if $x \in X$ and is defined as $\delta_{x\to y} R = \pi_{X\smallsetminus\{x\}\cup\{y\}}\, \mathsf{s}^x_y(\pi_{\mathsf{Att}} R)$;
- the *(set-theoretic) difference* $(-)$ and *union* $(\cup)$ operators can be applied to R, Q if $X = Y$;
- $1 = \{\langle\,\rangle\}$ (the singleton of the empty tuple) and $0 = \varnothing$ are included as constants (0-ary operations).

(Usually the presence of unary singleton relations (relations of the form $\{\langle x : a\rangle\}$ for some $x \in \mathsf{Att},\ u \in U$) is assumed; if so, 1 can be expressed as $\pi_\varnothing\{\langle x : a\rangle\}$, thus usually 1 is not included explicitely. We rather included 1 and omitted unary singletons.)

According to the heterogeneous algebra concept, we may introduce a set of variables for each relation sort $\subseteq \mathsf{Att}$ (however, if we do Codd algebra on $\mathcal{P}\left(U^{\subseteq \mathsf{Att}}\right)$, then the sorts are not disjoint: $\varnothing$ is conatined in each sort; this causes no problem); then it is clear what is meant by a *Codd term.* Now let t be a Codd term using $\mathsf{R}_1, \dots \mathsf{R}_n \in \tau$ as variables (R_i is taken as a variable of sort $\tau(\mathsf{R}_i)$). Then the *the query defined by* t is the mapping $\mathfrak{D} \mapsto t(R_1, \dots, R_n)$, where $\mathfrak{D}$ is a database (and the *generalized* query defined by t is the extension of the above mapping to arbitrary relational structures). All queries defined by Codd terms are (computable) queries in the sense of [AHV].

Recall that if $\mathfrak{A}, \mathfrak{B}$ are partial algebras of the same type, then a function $\xi : A \longrightarrow B$ is a *partial algebra homomorphism* if in case f is n-ary operation, and f is defined on $a_1, \dots, a_n \in \mathfrak{A}$, then f is also defined on $\xi(a_1), \dots, \xi(a_n) \in \mathfrak{B}$ and $\xi(f(a_1, \dots, a_n)) = f(\xi(a_1), \dots, \xi(a_n))$. □

Claim 1.16

i) Codd formulas and Codd terms define the same class of generalized queries (i.e., CFml *is* Codd algebra complete*).*

ii) d-mng$^-$ *is compositional on* CFml*, i.e. if $\mathfrak{M} = \langle U, I\rangle$ is a relational structure, then* d-mng$^-_{\mathfrak{M}}$ *is a* CFml $\longrightarrow \mathcal{P}\left(U^{\subseteq \mathsf{Att}}\right)$ *homomorphism (where this latter is to be supplied with the Codd terms corresponding to Codd connectives, as it is sketched in the proof of i)).*

iii) Codd algebra can be factorized with the null-values, i.e. the mapping

$$\mathcal{P}^*\left(U^{\subseteq \mathsf{Att}}\right) \longrightarrow \mathcal{P}\left(U^{\subseteq \mathsf{Att}}\right) : \quad R \mapsto \begin{cases} R & \text{if } \mathsf{R} \in \mathcal{P}\left(U^{\subseteq \mathsf{Att}}\right) \smallsetminus \{\varnothing\} \\ \varnothing & \text{if } R \text{ is of the form } \varnothing_X \end{cases}$$

is a homomorphism w.r.t. Codd algebra.

Proof.

i) There is a by-and-large correspondance between Codd connectives and Codd operations: $\mathbf{T}$ corresponds to 1, $\mathbf{F}$ to 0, $\wedge$ to $\bowtie$, existential quantifiers to projections, $\vee$ to $\cup$, $(\wedge\neg)$ to $-$, connectives of the form $(x = y \wedge\ )$ to selections, variable substitutions to renaming. The exact nature and technical problems of this correspondance and on this basis, a translation and a back-translation between Codd algebra and Codd formulas is overwieved in the next two paragraphs.

Using Definition 1.10 and 1.12, it is straightforward to give recursively a translation $\varphi_t \in \mathsf{CFml}$ to each Codd term t such that t and φ_t define the same generalized query. E.g. if we have φ_t and $\varphi_{t'}$, then the result of the query defined by $t \bowtie t'$ in a relational structure $\mathfrak{M}$ is $\mathsf{d\text{-}mng}_{\mathfrak{M}}(\varphi_t \wedge \varphi_{t'})$, thus $\varphi_{t \bowtie t'} \approx \varphi_t \wedge \varphi_{t'}$ is a correct choice. Finding the (derived) Codd connectives which exactly correspond to the other Codd operations, and other details are left to the reader.

Finding the translation $\varphi \mapsto t_\varphi$ is similar and is left to the reader. However, we emphasize here one difficulty: the Codd connective $(x = y \wedge\ )$ can be applied to $\varphi \in \mathsf{CFml}$ also in the cases $x \in \mathsf{frv}(\varphi) \not\ni y$ and $y \in \mathsf{frv}(\varphi) \not\ni x$, while the by-and-large corresponding Codd operation $\sigma_{x=y}$ can be applied to a relation(-variable) R of sort X only if $x, y \in X$. This can be helped in the following way: say, $x \in X \not\ni y$; then the basicly illegal selection $\sigma_{x=y}R$ can be taken as the abbreviation of the Codd term $\sigma_{x=y}(R \bowtie \delta_{x \to y}R)$. With this extension the operation $\sigma_{x=y}$ really corresponds to the connective $(x = y \wedge\ )$.

ii) and *iii)* is left to the reader. ∎

Corollary 1.17 *Any investigation concerning the expressive power of Codd formulas or queries can be referred to Codd's relational algebra.* ∎

Codd formulas (queries) have the advantage that they can be dircetly compared to other first-order formulas (queries). ***Hence in the sequel we do not use Codd algebra, rather we execute our investigations on Codd formulas. With this convention these investigations do not lose their relevance to the classical query language of Codd's relational algebra.***

Remark 1.18 (Codd's completeness) As it claimed in Claim 1.16 *i)*, Codd algebra can be interpreted into first-order logic, or as the phrase goes, *first-order logic is Codd algebra expressive*. However, we cannot state the converse without restrictions, since there are first-order queries which are not finite. Restrictions which enable us to give some kind of converse of the above statement can have either a syntactic or semantic nature.

A syntactic restriction is, e.g., considering only Codd formulas.

A semantic restriction is considering first-order queries with *active domain semantics*: in this sense the query defined by a formula φ is the mapping $\mathfrak{D} \mapsto \mathsf{d\text{-}mng}_{\mathfrak{M}_{\mathsf{adom}(\mathfrak{M})}}(\varphi)$ (where $\mathfrak{D}$ is database and the type is supposed to be finite in order to make $\mathsf{adom}(\mathfrak{M})$ definable by a formula). For domain independent formulas, like Codd formulas, the usual and active domain semantics coincide.

By Remark 1.12, Lemma 1.16 *i)* we have that under the active domain semantics every first-order query is expressible with a Codd algebra term, or as the phrase goes, *first-order queries under active domain semantics are Codd algebra bounded*. Comparing it with Codd algebra expressiveness, we can say that *first-order queries under active domain semantics are Codd algebra complete*. This is ***Codd's completeness theorem*** (see [Codd72]).

Those ones, who think that active domain is optimal, may wonder that our assumption of an infinite universe for databases is redundant: under the active domain semantics the whole complement of our finite active domain is totally irrelevant. A possible answer is the following:

Acitve domain semantics is usually considered in the following situation: an infinite set U_0 is fixed, the so-called *universal domain*, and we investigate the collection of databases with universe U_0. In this case every part of U_0 is made use of in some database, and does not occur a co-finite disposable part of U_0, as it would occur if we investigated only one database instance on U_0 with active domain semantics. See more on the importance of active domain semantics in Section 3. $\square$

2 The expressive power of relations on a subdomain

In database theory the universe is infinite but the relations "live" on finte subdomain of the universe. Hence when we investigate the expressive power of (first-order) queries, the following question arises: which relations and by which means can be expressed from these relations on a subdomain? This topic is discussed in this section, in order to apply these results in the next section.

We agree that $\mathsf{Att} = n$ for some non-negative integer n, as far as we do not break this agreement explicitely. (We will use von Neumann's convention by which a non-negative integer k is identified with the set $\{0, 1, \dots k-1\}$. However, instead of writing $i \in k$ we prefer writing $i < k$.)

Now consider the following situation: U is a set (the "universe"), and $D \subseteq U$ (the "subdomain"). An $R \subseteq U^n$ can be considered as a relation on D, if it is the *dummy representation* of a relation over D (w.r.t. U), i.e. for some $X \subseteq n,\ Q \subseteq D^X\ \ R = \pi_n Q$. If an element of $U \smallsetminus D$ occurs in such an R, then it must occur in a dummy attribute of R, thus $\pi_{\Delta R} R$ will surely do as Q; hence an equivalent condition to R is $\pi_{\Delta R} R \subseteq D^{\Delta R}$. Let ∂ stand for D^n. Then it is also equivalent if R is of the form $\mathsf{c}_{(X)}\, Q$ for some $Q \subseteq \partial$ and $X \subseteq n$. Finally, one more equivalent form: $(\exists X \subseteq n)(R = \mathsf{c}_{(X)}\, R \subseteq \mathsf{c}_{(X)}\, \partial)$. (The joy of checking the equivalence of these conditions is left to the reader.) Thus, we define the set of ***relations essentially on*** D as

$$\mathrm{RE}D \overset{\mathsf{def}}{=} \{\mathsf{c}_{(X)}\, Q : Q \subseteq \partial,\ X \subseteq n\}.$$

If D is a proper subset of U, dummy representation is a bijection between $\mathcal{P}\left(D^{\subseteq n}\right)$ and $\mathrm{RE}D$ (with inverse $R \mapsto \pi_{\Delta R} R$).

Let us pose the question: which relations on U are expressible from the relations on D, using first-order operations?

Generated subalgebra will be denoted by double brackets ($[\![\,\cdot\,]\!]$). Then the above question can be formalized as "How could we describe $[\![\mathrm{RE}D]\!] \leqslant \mathfrak{P}_{\mathsf{CA}}\left(U^n\right)$?" (if n runs to infinity, this questions covers the problem). We will show that $\mathrm{RE}D$ is a complete atomic Cs_n, thus its complete description consists of giving its ***atom structure***: the set of its atoms (which are to be considered as some kind of abstract "points", we may ignore their inner structure), the set of atoms included in the diagonals, and to each atom giving the set of those atoms which are included in its various cylinders.

The atom structure of a complete atomic CA_n can be considered as a relational structure $\mathfrak{At} = \langle At,\ \mathcal{C}_i(A, A'),\ \mathcal{D}_{ij}(A)\rangle_{i,j<n}$, where At is the set of atoms, $\mathcal{C}_i(A, A')$ says "$\mathsf{c}_i\, A$ contains A'", and $\mathcal{D}_{ij}(A)$ says "d_{ij} contains A".

Conversely, any atom structure (i.e., relational structure of the above type) induces a CA_n-type algebra ("*the complex algebra of the atom structure*"), as the above paragraph suggests (which happens to be a complete atomic Boolean algebra, on which c_i acts as a completely additive closure operator). Of course the complex algebra of the atom structure of a complete atomic $\mathfrak{A} \in \mathsf{CA}_n$ is isomorphic with $\mathfrak{A}$, and atom structures are isomorphic iff they induce isomorphic complex algebra — thus complete atomic CA_n's are really characterized by their atom structures. (More on atom structures and complex algebras can be found in [HMT1].)

Thus we give the atoms of $[\![\mathrm{RE}^* D]\!]$ as follows:

"$X \mathbin{\dot\cup} Y = n$" will stand for "$X, Y$ form a partition of n", i.e. $X \cup Y = n$ and $X \cap Y = \varnothing$. Recall that if g is a function, then $\mathsf{Ker}\, g$ is the following equivalence on its domain: $\langle i, j\rangle \in \mathsf{Ker}\, g \iff g(i) = g(j)$.

Assume $|U \smallsetminus D| = N$ (where N can also be an infinite cardinal). Now, if we have X, Y, f, ϑ such that $X \mathbin{\dot\cup} Y = n$, $f \in D^X$ and ϑ is an equivalence on Y such that it has no more than N blocks, then let

$$A^{DU}_{f\vartheta} = \left\{t \in U^n : t{\restriction}_X = f,\ \text{for } i \in Y \quad t_i \notin D,\ \vartheta = \mathsf{Ker}\left(t{\restriction}_Y\right)\right\}$$

(if D or U is understood from the context, we may omit them).

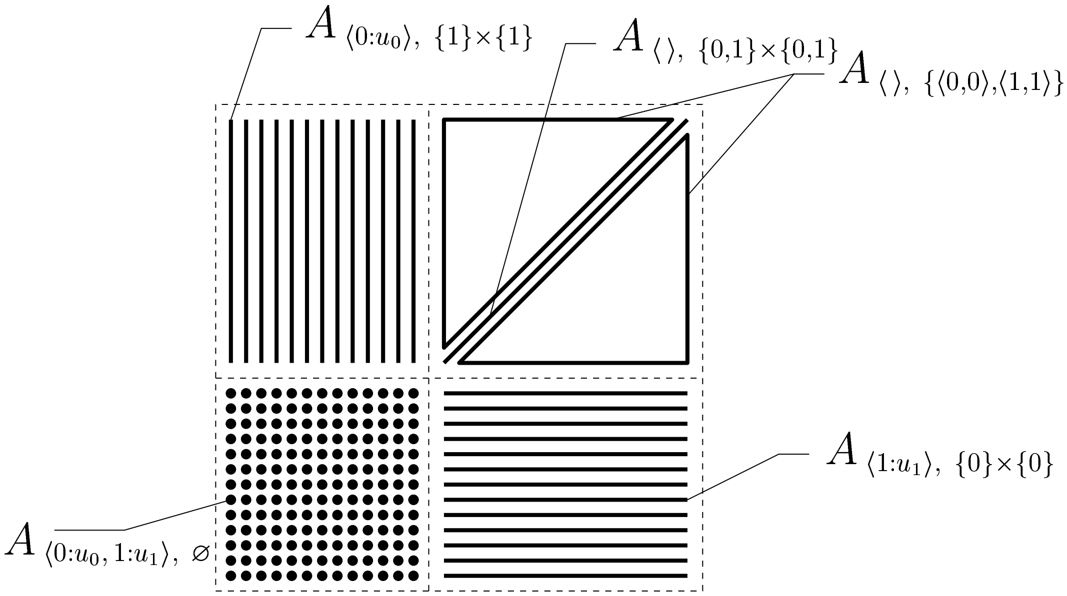
BA will serve as an abbreviation for "Boolean algebra". If $\mathfrak{A}$ is a CA_n and $X \subseteq \mathfrak{A}$, then $[\![X]\!]_{\mathsf{BA}}$ will denote the *Boolean generatum of* X, i.e. the generated subalgebra of X in the Boolean reduct of $\mathfrak{A}$. Moreover, any additional information conerning generated subalgebra is denoted as subscript of brackets, e.g. we write $[\![\mathrm{RE}^*D]\!]_U$ if we want to emphasize that we take generated subalgebra in $\mathfrak{P}_{\mathsf{CA}}(U^n)$. We will use $\mathrm{RE}^*D \overset{\mathsf{def}}{=} \mathrm{RE}D \cup \{\mathsf{d}_{ij} : i, j < n\}$.

Theorem 2.1 (The stable atom structure)

i) Let $D \subseteq U$, $N = |U \smallsetminus D|$. Then the following holds over U:
$[\![\mathrm{RE}D]\!] = [\![\mathrm{RE}^*D]\!]_{\mathsf{BA}}$, and it is a complete atomic Cs_n with atoms

$$At_U = \{A^U_{f\vartheta} : f \in D^X,\ \vartheta \text{ is an equivalence on } Y \text{ with } \#(\text{blocks}) \leqslant N$$
$$\text{for some } X, Y \text{ such that } X \mathbin{\dot\cup} Y = n\}.$$

ii) $[\![\mathrm{RE}D]\!]$ is stable at $\#(\text{outer elements}) \geqslant n$, i.e. if $U, U' \supseteq D$ and $|U \smallsetminus D|$, $|U' \smallsetminus D| \geqslant n$, then $[\![\mathrm{RE}D]\!]_U \cong [\![\mathrm{RE}D]\!]_{U'}$; more concretely, $A^U_{f\vartheta} \mapsto A^{U'}_{f\vartheta}$ (with f, ϑ as above) is an isomorphism between the atom structures of $[\![\mathrm{RE}D]\!]_U$ and $[\![\mathrm{RE}D]\!]_{U'}$.
The stable atom structure is illustrated by the following figure for $n = 2$:



Proof.
i) Consider that $1 = U^n \in \mathrm{RE}^*D$. Hence to see that $[\![\mathrm{RE}^*D]\!]_{\mathsf{BA}}$ is atomic and its set of atoms is exactly $At = At_U$, we need the following:

- $\varnothing \notin At$ (this will be left to the reader);
- an $R \in \mathrm{RE}^* D$ is the union of some (possibly infinitely many) elements of At;
- if A, A' are distinct elements of At, then they can be separated by some $R \in \mathrm{RE}^* D$ (i.e., $A \subseteq R$ and $A' \cap R = \varnothing$, or vica cersa).

We introduce the following convention up to this proof: whenever we write X, Y, f, ϑ it is understood that $n = X \mathbin{\dot\cup} Y$, $f \in D^X$, ϑ is an equivalence on Y with $\#(\text{blocks}) \leqslant N$; and similarly for the comma versions X', Y', f', ϑ'.

Producing $R \in \mathrm{RE}^* D$ from At:

- $R = \mathsf{d}_{ij}$: $\mathsf{d}_{ij} = \bigcup\{A_{f\vartheta} : i, j \in X \text{ and } f_i = f_j\} \cup \bigcup\{A_{f\vartheta} : i, j \in Y \text{ and } \langle i, j\rangle \in \vartheta\}$;
- $R = \pi_n Q$, where $Q \subseteq D^Z$, $Z \subseteq n$: $R = \bigcup\left\{A_{f\vartheta} : Z \subseteq X \text{ and } f\restriction_Z \in Q\right\}$.

Now, if $A_{f\vartheta}$ and $A_{f'\vartheta'}$ are distinct elements of At, then how to separate them?

- If $X \neq X'$, then, say, $X \ni i \notin X'$. In this case $A_{f\vartheta} \subseteq \mathsf{c}_i^{\perp}\,\partial\,(= \pi_n D^{\{i\}})$ and $A_{f'\vartheta'} \cap \mathsf{c}_i^{\perp}\,\partial = \varnothing$.
- If f and f' are not coherent, then, say, $i \in X \cap X'$ and $f_i \neq g_i$. In this case $A_{f\vartheta} \subseteq \pi_n\{f\}$ and $A_{f'\vartheta'} \cap \pi_n\{f\} = \varnothing$.
- If $f = f'$, then it must be that $\vartheta \neq \vartheta'$, so say, $i, j \in Y$ and $\vartheta \ni \langle i, j\rangle \notin \vartheta'$. In this case $A_{f\vartheta} \subseteq \mathsf{d}_{ij}$ and $A_{f'\vartheta'} \cap \mathsf{d}_{ij} = \varnothing$.

Now we turn to completeness. If $X \mathbin{\dot\cup} Y = n$, ϑ is an equivalence on Y with $(\leqslant N)$ blocks, then let $A_{X\vartheta} = \bigcup_{f \in D^X} A_{f\vartheta}$. As it easy to see, the sets $A_{X\vartheta}$ form a finite partition of U^n, and

$$A_{X\vartheta} = \mathsf{c}_{(X)}^{\perp}\,\partial \smallsetminus \left(\bigcup_{i \in Y} \mathsf{c}_i^{\perp}\,\partial\right) \cap \left(\bigcap_{i,j \in Y} \varepsilon_{ij}\mathsf{d}_{ij}\right),$$

where $\varepsilon_{ij} = \begin{cases} \text{the empty word} & \text{if } \langle i, j\rangle \in \vartheta \\ \text{“}\smallsetminus\text{”} & \text{if } \langle i, j\rangle \notin \vartheta \end{cases}$; thus $A_{X\vartheta} \in [\![\mathrm{RE}^* D]\!]_{\mathsf{BA}}$. Hence for completeness it is enough to see that if $Q \subseteq D^X$, then $\bigcup_{f \in Q} A_{f\vartheta} \in [\![\mathrm{RE}^* D]\!]_{\mathsf{BA}}$. And this is so: $\bigcup_{f \in Q} A_{f\vartheta} = \pi_n Q \cap A_{X\vartheta}$.

We have seen yet which atoms belong to d_{ij} (thus $\mathcal{D}_{ij}\,(A_{f\vartheta}) \iff (i, j \in X$ and $f_i = f_j)$ or $(i, j \in Y$ and $\langle i, j\rangle \in \vartheta$).) The only component of an atom structure which we have not concerned yet is the relation $\mathcal{C}_i(A, A')$. For $i < n$ let

$$\mathcal{C}_i\,(A_{f\vartheta}, A_{f'\vartheta'}) \overset{\mathsf{def}}{\iff} X - i = X' - i,\ f\restriction_{X-i} = f'\restriction_{X-i}, \text{ and } \vartheta\restriction_{Y-i} = \vartheta'\restriction_{Y-i}$$

(where $X - i$ abbreviates $X \smallsetminus \{i\}$, etc.).

Checking that this $\mathcal{C}_i$ is the proper one, i.e. for $A \in At$ $\ \mathsf{c}_i\, A = \bigcup_{A' \in At} \mathcal{C}_i(A, A')$.

$\mathsf{c}_i\, A \supseteq A' \iff \mathsf{c}_i\, A \cap A' \neq \varnothing \iff \mathcal{C}_i(A, A')$, is left to the reader. This fact implies that the complete BA generated by At is closed w.r.t. cylindrifications, i.e. $[\![\mathrm{RE}^* D]\!]_{\mathsf{BA}} = [\![\mathrm{RE} D]\!]$.

ii) The set of atoms At_U for any $U \supseteq D$ is given via the parameters f, ϑ, just like the $\mathcal{C}_i$, $\mathcal{D}_{ij}$ relations on At_U. For U's satisfying $|U \smallsetminus D| \geqslant n$ “ϑ has no more than n

blocks" is not a restriction, so the above entities are given by the same set of parameters (and the same way). Hence for such big U, U' $A^U_{f\vartheta} \mapsto A^{U'}_{f\vartheta}$ is an isomorphism between the atom structures of $[\![\mathrm{RE}D]\!]_U$ and $[\![\mathrm{RE}D]\!]_{U'}$. ■

As it is done usually, from now on Att will refer to an infinite collection of attribute.

Here we must stop for making some subtle distinction. Let us use the notation $[\![\mathrm{RE}D]\!]_n$, indicating the specific n we choose for Att ("the dimension"). We feel like thinking of $[\![\mathrm{RE}D]\!]_n$ as the set of those n-ary relations on U, which are first-order expressible from relations over D, however, we connot do so without further considerations: if we are precise, we can only say, $[\![\mathrm{RE}D]\!]_n$ is the set of those n-ary relations on U, which are first-order expressible from n-ary relations over D ("expressible using only n variables"). These further considerations are included in the following claim, which also sheds some light on the role of the atoms $A_{f\vartheta}$:

Claim 2.2 *Let $\mathfrak{M} = \langle U, I\rangle$ be a relational structure with $D = \mathsf{adom}(\mathfrak{M})$. Then any automorphism-invariant n-ary relation on U (w.r.t. $\mathfrak{M}$) is in $[\![\mathrm{RE}D]\!]_n$.*

Especially $[\![\mathrm{RE}D]\!]$ is the set of those n-ary relations on U, which are first-order expressible from relations over D.

Proof. If f is function from some set A to some set B, then $f^{[n]} : A^n \longrightarrow B^n$ will denote its n-th direct power, i.e. $f^{[n]}(a_1, \ldots, a_n) = (f(a_1), \ldots, f(a_n))$. If G is a group of permutations, then let $G^{[n]} = \{g^{[n]} : g \in G\}$ — this is also a permutation group, and it is isomorphic with G.

Now let $\mathfrak{M} = \langle U, I\rangle$ be a relational structure with active domain D, and G be its automorphism group. Besides let H be group of those permutations of U, which leave D elementwise fixed. Since $H \leqslant G$ and for $Q \subseteq U^n$ and $\alpha \in G$ the earlier defined $\alpha(Q)$ and $\alpha^{[n]}(Q)$ are the same, any automorphism-invariant n-ary relation on U is fixed by $H^{[n]}$.

Generally, if a group consist of permutations of some set S, then a subset of S is fixed by the group iff it respects the orbits of the group, i.e., if it is the union of some orbits. And it is easy to see that *the orbits of $H^{[n]}$ are exactly the atoms $A^{DU}_{f\vartheta}$*. Hence we have: an automorphism-invariant n-ary relation on U is the union of some atoms $A^{DU}_{f\vartheta}$. I.e., it is in $[\![\mathrm{RE}D]\!]_n$. ■

Now we list some corollaries of Theorem 2.1:

The "$[\![\mathrm{RE}D]\!] = [\![\mathrm{RE}^*D]\!]_{\mathsf{BA}}$" part of the Theorem 2.1 can be stated in a more classical-logic tasted way (even without using Claim 2.2):

Corollary 2.3 *If $\mathfrak{M} = \langle U, I\rangle$ is a relational structure of type τ such that for some $D \subseteq U$ the finitary relations on D are exactly the interpretations of the relation symbols of τ (by I), then the theory of $\mathfrak{M}$ is quantifier-eliminable.* ■

Remark 2.4 Theorem 2.1 generalizes [HuSu, Lemma 4.2.]:

Let $\varphi(x_0, \ldots, x_{n-1}) \in \mathsf{Fml}$, $\mathfrak{M}_0 = \langle U_0, I\rangle$ be a relational structure such that with $D = \mathsf{adom}_{\mathfrak{M}_0}(\varphi)$ we have $|U_0 \smallsetminus D| = n$. (Identifying the variable x_i with i and translated to our teminology,) [HuSu, Lemma 4.2.] claims that if we take $U \supseteq U_0$, $\mathfrak{M} = \langle U, I\rangle$ (here I is still the same!), then $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}_0}(\varphi) = \mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi) \cap U_0^n$.

By Theorem 2.1 if $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}_0}(\varphi) = \bigcup_i A^{U_0}_{f_i\vartheta_i}$, then $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi) = \bigcup_i A^{U}_{f_i\vartheta_i}$, and for each f, ϑ $A^{U_0}_{f\vartheta} = A^U_{f\vartheta} \cap U_0^n$ — from these immediately follows the above cited claim.

However, by these considerations we are not only able to determine $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}_0}(\varphi)$ from $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi)$, but we are also able to determine $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi)$ from $\mathsf{d\text{-}mng}_{\mathfrak{M}_0}(\varphi)$. Thus if D is finite, we can use $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}_0}(\varphi)$ as an encoding of $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi)$ for any $\mathfrak{M} = \langle U, I\rangle$ with $U \supseteq D$ big enough. This idea is elaborated in Section 4. □

Generally speaking, from theorem 2.1 follows that structures with finite active domain behave very similar to finite ones. Here we give a concrete example of this:

It is a well-known fact that if $\mathfrak{M} = \langle U, I\rangle$ is a *finite* relational structure, then the converse of automorphism invariance also holds: *any automorphism-invariant relation is first-order expressible* — since if $U = \{u_1, \dots, u_m\}$, and Q is an automorphism-invariant relation on U (with attributes $x_1, \dots, x_n$), then (provided the type τ has finitely many relation symbols) Q is expressed by the formula

$$\exists y_1, \dots, y_m \Bigg(\bigwedge_{i \neq j} y_i \neq y_j \wedge \bigwedge \Big\{ \mathsf{R}\left(y_{i_1}, \dots y_{i_{\tau(\mathsf{R})}}\right) : \mathsf{R} \in \tau,\ \left\langle u_{i_1}, \dots, u_{i_{\tau(\mathsf{R})}} \right\rangle \in R \Big\} \wedge$$
$$\wedge \bigvee \{x_1 = y_{i_1} \wedge \dots \wedge x_n = y_{i_n} : \langle u_{i_1}, \dots, u_{i_n}\rangle \in Q\} \Bigg). \qquad (*)$$

— This formula says "for some $y_1, \dots, y_m$ the mapping $u_i \mapsto y_i$ is an automorphism of $\mathfrak{M}$, and the tuple $\langle x_1, \dots, x_n\rangle$ is in the image of Q by this automorphism". (Besides we may assume τ is finite: otherwise, as it is easy to see, $\mathfrak{A}$ has a finite-type reduct with the same set of automorphisms.)

As it follows from Claim 2.2, this obsevation can be generalized in the following way:

Corollary 2.5 *If a relation structure has finite active domain, then any automorphism-invariant relation on its universe is first-order expressible.*

Proof. Let $\mathfrak{M} = \langle U, I\rangle$ be a relational structure with a finite acitve domain D. By Claim 2.2, any automorphism-invariant n-ary relation on U is the union of some atoms $A_{f\vartheta} = A^{DU}_{f\vartheta}$.

Thus if $Q \subseteq U^n$ is automorphism-invariant, and $D = \{u_1, \dots, u_m\}$, Q will expressed by a slightly modified form of $(*)$: the disjunction part of the formula is to be replaced with $\bigvee\{\alpha_{f\vartheta} : A_{f\vartheta} \subseteq Q\}$, where, if $f = \langle i_1 : u_{j_1}, \dots, i_k : u_{j_k}\rangle$ with some $n \supseteq \{i_1, \dots i_k\} = X,\ 1 \leqslant j_1, \dots, j_k \leqslant m$, then, using $n = X \dot\cup Y$, $\alpha_{f\vartheta}$ is the formula

$$x_{i_1} = y_{j_1} \wedge \dots \wedge x_{i_k} = y_{j_k} \wedge \bigwedge_{\langle i,i'\rangle \in \vartheta} x_i = x_{i'} \wedge \bigwedge_{\langle i,i'\rangle \in (Y \times Y) \smallsetminus \vartheta} x_i \neq x_{i'}.$$

■

Corollary 2.6 *If $\mathfrak{M} = \langle U, I\rangle$ is a relational structure such that $D = \mathsf{adom}(\mathfrak{M})$ is finite, then the relations on D which are first-order expressible in $\mathfrak{M}$, and the relations expressible in $\mathfrak{M}{\restriction}_D$ are the same.*

Proof. This statement immediately follows from Corollary 2.5 and the fact that $\mathsf{Aut}(\mathfrak{M}) = \mathsf{Aut}(\mathfrak{M}_\delta) \times \mathsf{Sym}(U \smallsetminus D)$ (where Aut means "the automorphism group of", and Sym means "the group of all permutations of"). ■

This observation is highly generalized in the following theorem; thus the reason for including this corollary was its relatively simple semantic proof. The following theorem operates by syntactic means — not too much complex ones, but ones which are difficult to communicate precisely *and* perpicuously. In the following theorem we will not assume the finiteness of the active domain, and we will execute the process of expressing simutaneously in many relational structures: a formula will be transcribed in such a way that the original formula in $\mathfrak{M}$ and the trascribed formula expresses the same relation on D, and this transcription can be the same in a broad class of relational structures. Thus the following Theorem claims a "global" statement (whereas Theorem 2.1 and its corollaries claim "local" statements: there is only one structure considered at a moment).

Theorem 2.7 (Simulating formulas in substructures) *Let $\delta(x)$ be a Codd formula. If $\mathfrak{M} = \langle U, I\rangle$ is a relational structure, let $D_{\mathfrak{M}} \subseteq U$ be $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}\,\delta$ (here we used a $U^{\{x\}} = U$ identification).*

For a cardinal N and a formula φ let $\mathcal{K}_N(\varphi)$ be the class of those structures $\mathfrak{M} = \langle U, I\rangle$ in which $|U \smallsetminus D_{\mathfrak{M}}| = N$, and $\mathsf{adom}_{\mathfrak{M}}(\varphi) \subseteq D_{\mathfrak{M}}$.

There exists a formula φ'_N such that it is equivalent with φ in $\mathcal{K}_N(\varphi)$ and for all $\mathfrak{M} \in \mathcal{K}_N(\varphi)$, $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi) \cap (D_{\mathfrak{M}})^{\mathsf{frv}(\varphi)} = \mathsf{d\text{-}mng}^{-}_{\mathfrak{M}_\delta}\,\varphi'_N$. For N's bigger than the number of variables in φ, the formulas φ'_N can be the same formula φ'.

Now let us fix such a $\delta(x)$, φ.

To prove this theorem, first we introduce the ***δ-restricted quantifiers*** $\exists_\delta$, $\forall_\delta$: if φ is a formula, $x \in \mathsf{Att}$, let $\exists_\delta\,\varphi \approx \exists x(\delta(x) \wedge \varphi)$, $\forall_\delta\,\varphi \approx \forall x(\delta(x) \to \varphi)$. We say a formula is ***δ-restricted*** if it is built up from atomic formulas and δ by the Boolean connectives and δ-restricted quantifiers. (Thus we treat δ as if it were a relation symbol).

Lemma 2.8 *φ is equivalent in $\mathcal{K}_N$ with a δ-restricted formula φ'_N. For N's bigger than the number of variables in φ the formulas φ'_N can be the same formula φ'.*

Proof. Let us fix N. In this proof we will use the notation "$\xi \equiv \eta$" for "ξ and η are non-typed equivalent in $\mathcal{K}_N(\varphi)$". We will use the word "equivalent" also in this sense.

We prove by formula induction; what makes this possible is the obsevation that if ψ is a subformula of φ, then $\mathcal{K}_N(\varphi) \subseteq \mathcal{K}_N(\psi)$.

The only non-trivial step is to show that if $\varphi \approx Qx\,\psi$, where ψ is δ-restricted and Q is some of $\exists$, $\forall$, then φ is typed equivalent in $\mathcal{K}_N(\varphi)$ with some δ-restricted formula φ'. We will perform this step with $Q = \exists$.

First of all, we may assume that there are no duplicated quantifiers, i.e. no subformula of φ of the form $Q_1\,y\,\xi$ has proper subformula of the form $Q_2\,y\,\eta$ (where y is some (†)
variable, Q_1, Q_2 are quantifiers). (Otherwise an appropriate substitution of variables helps this condition).

Let $x_1, \ldots, x_n$ be those variables which occur in ψ in some subformula of the form $x = x_i$ (or $x_i = x$, further on we will not distinguish between these two) such that *this* occurrence of x_i is not bounded by a quantifier. We define the *label formulas* as follows:

- $\delta(x)$ is a label formula;

- for any choice of $\varepsilon_1, \ldots, \varepsilon_n$ such that ε_i is either the symbol $\neg$ or the empty word, $\neg\delta(x) \wedge \bigwedge_{i=1}^{n} \varepsilon_i\,(x = x_i)$ is a label formula.

The set of label formulas will be denoted by Λ. Then $\bigvee_{\lambda\in\Lambda} \lambda \equiv \mathbf{T}$, hence $\varphi \approx \exists x\, \psi \equiv \bigvee_{\lambda\in\Lambda} \exists x\,(\lambda \wedge \psi)$.

We will get φ' by transforming each disjunction member $\exists x\,(\lambda \wedge \psi)$ to a δ-restricted form.

For $\lambda \approx \delta(x)$ there is nothing to do: this disjunction member is simply $\exists_\delta\, \psi$. This formula has the same free variables as $\varphi \approx \exists x\, \psi$, hence the *typed* equivalence of φ and φ'.

Now let λ be a label formula of the form $\neg\delta(x) \wedge \bigwedge_{i=1}^{n} \varepsilon_i\,(x = x_i)$. Let ψ' be the formula which we get if we replace the subformulas of ψ of the form $x = x_i$ with $\varepsilon_i\,\mathbf{T}$ and any other atomic subformula of ψ, in which x occurs, with $\mathbf{F}$ (hence x does not occur in ψ'). We claim that $\lambda \wedge \psi \equiv \lambda \wedge \psi'$. If we have this, we are ready, because then $\exists x\,(\lambda \wedge \psi) \equiv \exists x\lambda \wedge \psi'$, and it is easy to rewrite $\exists x\lambda$ to an equivalent form λ_N which is a Boolean combination of formulas of the form $\delta(x_i)$, $x_i = x_j$. (This is the only point in the construction, where we find a dependece on N, and only if for all i $\varepsilon_i = \neg$, and just up to a point: if all ε_i's are "$\neg$" and $N > n$, then $\lambda_N \approx \mathbf{T}$ will do.)

To get ψ' from ψ by equivalence-preserving transformations, we will proceed through several steps.

For any formula $\vartheta(y)$, ξ, η the following transformation rules are valid (even with logical equivalence):

- $\vartheta \wedge \neg\xi \equiv \neg(\vartheta \to \xi)$ and $\vartheta \to \neg\xi \equiv \neg(\vartheta \wedge \xi)$;
- for $* \in \{\wedge, \vee\}$, $\vartheta \wedge (\xi * \eta) \equiv (\vartheta \wedge \xi) * (\vartheta \wedge \eta)$ and $\vartheta \to (\xi * \eta) \equiv (\vartheta \to \xi) * (\vartheta \to \eta)$;
- if z is a variable other than y, and Q is one of $\exists$, $\forall$, then $\vartheta \wedge Q_\delta\, z\, \xi \equiv Q_\delta\, z(\vartheta \wedge \xi)$ and $\vartheta \to Q_\delta\, z\, \xi \equiv Q_\delta\, z(\vartheta \to \xi)$.

By the notation $\vartheta(\xi_1, \ldots, \xi_k)$ we will mean that the ξ_i-s are subformulas of ϑ, and ϑ is produced from them by the connectives (e.g., if $\vartheta \approx \xi \wedge (\xi \to \eta)$, then we may emphasize the structure of ϑ by writing $\vartheta \approx \$\vartheta(\xi,\, \xi \to \eta)$). When using this notation, we may apply the usual conventions for substitution (e.g., following the above example, $\vartheta(\xi,\, \eta)$ abbreviates $\xi \wedge \eta$).

Now we show that if $Q_\delta\, x_i\, \xi$ is a subformula of φ (where $1 \leqslant i \leqslant n$), and $x{=}x_i$ is a subformula of $Q_\delta\, x_i\, \xi$, then by replacing $x{=}x_i$ with $\delta(x_i) * (x{=}x_i)$ in $Q_\delta\, x_i\, \xi$ (where $*$ is the appropriate one of $\wedge$, $\to$) yields an equivalent formula to $Q_\delta\, x_i\, \xi$. We will do it for $Q = \exists$, for $Q = \forall$ the proof goes simlarly.

ξ is of the form $\xi(x{=}x_i, \eta_1, \ldots, \eta_k)$. $\exists_\delta x_i\, \xi \approx \exists x_i(\delta(x_i) \wedge \xi)$, and by the above transformation rules and (†) $\delta(x_i)$ can be "pushed into ξ", by which we get

$$\exists x_i(\delta(x_i) \wedge \xi) \equiv \exists x_i\, \xi\Big(\delta(x_i) * (x{=}x_i),\ \delta(x_i) *_1 \eta_1, \ldots, \delta(x_i) *_k \eta_k\Big),$$

where $*$, $*_1, \ldots *_k$ are are some of $\wedge$, $\to$. Since $\delta(x_i) * (x{=}x_i) \equiv \delta(x_i) * \big(\delta(x_i) * (x{=}x_i)\big)$, we have

$$\exists x_i\Big(\delta(x_i) * (x{=}x_i),\ \delta(x_i) *_1 \eta_1, \ldots, \delta(x_i) *_k \eta_k\Big) \equiv$$

$$\equiv \exists x_i\Big(\delta(x_i) * \big(\delta(x_i) * (x{=}x_i)\big),\ \delta(x_i) *_1 \eta_1, \ldots, \delta(x_i) *_k \eta_k\Big),$$

and, by "pulling out $\delta(x_i)$ from ξ" we get the equivalent formula $\exists x_i\Big(\delta(x_i) \wedge \xi(\delta(x_i) * (x{=}x_i),\, \eta_1, \ldots, \eta_k\Big) \approx \exists_\delta x_i\, \xi\Big(\delta(x_i) * (x{=}x_i), \eta_1, \ldots, \eta_k\Big)$. Thus we have

$$\exists_\delta x_i\, \xi \approx \exists_\delta x_i\, \xi(x{=}x_i,\, \eta_1, \ldots, \eta_k) \equiv \exists_\delta x_i\, \xi\Big(\delta(x_i) * (x{=}x_i), \eta_1, \ldots, \eta_k\Big).$$

By doing this replacement systematically, we get that $\psi \approx \psi(\alpha_1, \ldots, \alpha_k)$ (where the α_j's are atomic) can be transformed to a formula $\psi_1 \approx \psi(\alpha_1^1, \ldots, \alpha_k^1)$, where the α_j^1-are either atomic formulas, or formulas of the form $\delta(x_i){*}(x{=}x_i)$, where $* \in \{\wedge, \to\}$, and if α_j is of the form $x = x_i$, then this occurrence of x_i is free.

Now consider $\neg\delta(x) \wedge \psi_1$: similarly as we did, we push $\neg\delta(x)$ into ψ_1, getting a formula $\psi(\alpha_1^2, \ldots, \alpha_k^2)$, where the α_j^2's are of the form $\neg\delta(x) * \alpha_j^1$ with α_j^1 atomic, or are of the form $\neg\delta(x) * \big(\delta(x) \diamond (x{=}x_i)\big)$, where $*$, $\diamond$ are one of $\wedge$, $\to$ (and if α_j is of the form $x = x_i$, then this occurrence of x_i is free).

Now we "execute" many of the α_2^j's:

- if α_2^j is of the form $\neg\delta(x) * \mathsf{R}(x, \vec{y})$, then, since $\mathcal{K}_N(\varphi) \models \mathsf{adom}_R(x) \to \delta(x)$, it is equivalent with $\neg\delta(x) * \mathbf{F}$;
- if α_2^j is of the form $\neg\delta(x) * \big(\delta(x) \diamond (x{=}x_i)\big)$, then it is equivalent with $\neg\delta(x) * (\delta(x) \diamond \mathbf{F})$ (checking this is left to the reader).

Replacing these α_2^j's by these equivalents, we get some formula ψ_3; then pulling out $\neg\delta(x)$ from ψ_3, and $\delta(x)$ from the subformulas $Q_\delta\, x_i\, \xi$, we get ψ_4, which satisfies $\neg\delta(x) \wedge \psi \equiv \neg\delta(x) \wedge \psi_4$, and this ψ_4 is produced from ψ by replacing those α atoms, in which x occurs, with $\mathbf{F}$, *except for* if α is of the form $x = x_i$ and this occurence of x_i is free.

Thus ψ_4 is of the form $\psi_4 \approx \psi_4(\vartheta_1, \ldots, \vartheta_l)$, where the ϑ_j's can either be atoms, or subformulas of the form $Q_\delta\, x_i\, \xi$, in which x does not occur, or can be atoms of the form $x = x_i$, and this occurrence of x_i is free.

So consider the formula $\varepsilon_1\,(x{=}x_1) \wedge \psi_4(\vartheta_1, \ldots, \vartheta_m)$: again we push $\varepsilon_1\,(x = x_1)$ into ψ_4, to the ϑ_j's — we can do it because of (†) and the above description of the ϑ_j's. Hence we have an equivalent formula $\psi_4\big(\varepsilon_1\,(x{=}x_1) *_1 \vartheta_1, \ldots, \varepsilon_1\,(x{=}x_1) *_m \vartheta_m\big)$, where $*_j \in \{\wedge, \to\}$. Now if for some j, $\vartheta_j \approx x = x_1$, then $\varepsilon_1\,(x{=}x_1) *_j \vartheta_j \equiv \varepsilon_1\,(x{=}x_1) *_j \varepsilon_1\,\mathbf{T}$. We execute a replacement acccording to this equivalence, then pull out $\varepsilon_1\,(x = x_1)$.

Repeating this procedure for all $i = 2, \ldots, n$, we get a ψ_5 which satisfies $\lambda \wedge \psi \equiv \lambda \wedge \psi_5$; and this ψ_5 is just the same as ψ'. ∎

Proof (of Theorem 2.7). Let $\mathfrak{M} = \langle U, I\rangle$ be a relational structure such that $\mathsf{adom}_\mathfrak{M}(\varphi) \subseteq D_\mathfrak{M}$. By Lemma 2.8 it is enough to show that if φ is δ-restricted, then $\mathsf{d\text{-}mng}^{-}_{\mathfrak{M}}(\varphi) \cap D_\mathfrak{M}^{\mathsf{frv}(\varphi)} = \mathsf{d\text{-}mng}^{-}_{\mathfrak{M}_\delta}\,\varphi$ (in this proof we will call this property "restrictable").

If φ is atomic, or if $\varphi \approx \delta$, then it is surely restrictable.

We have to show that if the some formulas are restrictable, then their Boolean combinations are also restrictable, and applying δ-restricted quantifiers on them, we get restrictable formulas. This is left to the reader. ∎

Remark 2.9 Theorem 2.7 was proved in [HuSu, Lemma 4.5]; we lent the idea of the proof from this location, however, their proof was unnecessarily complicated: instead of transforming a formula to a δ-restricted one step-by-step, recursively, they showed a highly complex construction for transforming formulas directly to δ-restricted ones. They claimed the equivalence of the transformed formula in [HuSu, Lemma 4.5], but they omitted the verification, which could be essentially the same as our proof for Lemma 2.8.

Moreover, they claim that their transformed formula is with length at most $O(n!) \approx 2^{O(n \log n)}$ in the size of the original formula, whilst, as it is easy to see, our transformed formula is with length at most $O(2^n)$ in the size of the original formula. $\square$

3 Computability, domain independence, finiteness

In the early period of database theory there was one canonized query language (Codd's relational algebra), and the notion of query was formalized as a mapping which can be described in *relational complete languages* (i.e., ones which are equivalent with Codd's algebra in expressive power) (see 1.18, [Codd72]).

However, in an intuitive sense such mappings, like finding the transitive closure of a binary relation, or determining the parity of a unary relation, were also considered as queries, although they are not expressible in relational complete languages. Hence the question has been posed: what is the broadest reasonable extension of the notion of "query"? And the commonly accepted answer was the notion of *computable query*: a mapping which has a finite collection of relations ,,given by tables" (i.e., finite ones) as input, a single relation table as output, which is computable, and which is *generic*, i.e. it does not depend on the actual representation of data, and it does not use any "meta-information" which is not given in the input database (e.g. a total order of the universe (maybe derived from an enumeration)) (see [ChaHa]); we give this conditions more formally.

First of all we fix a *finite* relational type τ, and an infinite recursive set, U_0, called *the universal domain.* All considered databases will have U_0 as their universe. This agreement is called ***the universal domain assumption***, and it will stay valid until we declare it unvalid. The word "database" will be abbreviated as "*db*". We will use the symbol $\cong$ to refer to isomorphism of db's.

Definition 3.1 A ***quasi-query*** is a mapping on db's which associates to each db a relation on U_0 with finite arity. (We can also speak of *partial quasi-queries* in a similar manner, we will not digress on partial versions explicitely if it is not necessary). A quasi-query q is ***well-typed*** if there is an $X \subseteq \mathsf{Att}$ such that all the values of q are relations of sort X. A quasi-query is ***finite*** if all of its values are finite relations. A quasi-query q is ***generic***, or *has the isomorphism invariance property* if for any $f : \mathfrak{D} \longrightarrow \mathfrak{D}'$, where $\mathfrak{D}$, $\mathfrak{D}'$ are db's and f is an isomorphism between them, $q(\mathfrak{D}') = f(q(\mathfrak{D}))$ (in case of a generic *partial* quasi-query q, if $\mathfrak{D} \cong \mathfrak{D}'$, then q is to be simultaneously defined or undefined on $\mathfrak{D}$, $\mathfrak{D}'$).

A ***(computable) query*** is a *recursive* well-typed finite generic quasi-query (which makes sense if we fix some recursive encoding of database and relation instances on U_0). $\square$

Remark 3.2 (The finiteness principle) The following idea will be called ***the finiteness principle of database theory:***

"We are willing to consider a quasi-query as a computable one only if it is finite."

Hence the first-order queries in general are not total computable queries; they can be taken partial ones if we declare them undefined on such inputs, where they had an infinite output. $\square$

In this section, the word query will be used in the following senses:

- we may use the expressions "first-order query" for quasi-queries defined by first-order formulas, and "the query φ" (where $\varphi \in \mathsf{Fml}$) in the sense of Section 1;

- we may speak of "computable queries" (either total or partial); if we use the word "query" alone, then we mean "computable query".

The notions of recusiveness, well-typedness, finiteness are quite clear; but how to imagine genericity? For a given db $\mathfrak{D}$, which relations on U_0 are of the form $q(\mathfrak{D})$, for some generic quasi-query q? It is easy to see that exactly the automorphism-invariant ones (w.r.t. $\mathfrak{D}$), thus, by Corollary 2.5, exactly the first-order expressible ones. Hence

Claim 3.3 *A quasi-query q is* generic *iff for each isomorphism class $\mathcal{C}$ of db's there is a formula φ such that $q\lceil_{\mathcal{C}} = \big(\mathsf{d\text{-}mng}^{-}(\varphi)\big)\lceil_{\mathcal{C}}$.* ∎

And how could we describe *finite* generic quasi-queries? This means describing finite automorphism-invariant relations (w.r.t. some db $\mathfrak{D}$). We can give an even more elementary characterization of such relations, i.e., we may avoid referring to Theorem 2.1 (in the previous characterization we referred to Theorem 2.1 by using Corollary 2.5):

If for some finite $X \subseteq \mathsf{Att}$, $R \subseteq U_0^X$ is finite, and it is automorphism-invariant w.r.t. some db $\mathfrak{D}$, then, as we found it in the proof of Claim 1.8, $R \subseteq \mathsf{adom}(\mathfrak{D})^X$, and then by Corollary 2.5, it is also expressible in $\mathfrak{D}_{\mathsf{adom}(\mathfrak{D})}$ by some formula φ, and then by Remark 1.12, the Codd-formula $\varphi\lceil_{\mathsf{adom}(\mathfrak{D})}$ expresses R in $\mathfrak{D}$. On the other hand, it is clear that relations expressed by Codd formulas are automorphism-invariant and finite (cf. Claim 1.11). Thus

Claim 3.4 (Bancilhon, Paredaens) *A relation is expressible in a given db $\mathfrak{D}$ by Codd formulas if it is finite and automorphism-invariant w.r.t. $\mathfrak{D}$.* ∎

— Cf. [Ba], [Par]. Hence

Corollary 3.5 *A quasi-query is* finite generic *iff for each isomorphism class $\mathcal{C}$ of db's there is a Codd formula φ such that $q\lceil_{\mathcal{C}} = \big(\mathsf{d\text{-}mng}^{-}(\varphi)\big)\lceil_{\mathcal{C}}$.* ∎

We can give further logical characterizations of properties of quasi-queries:

Claim 3.6

i) A quasi-query q is well-typed generic *iff it is locally first-order, i.e. for each finite set $\mathcal{D}$ of db's there is a formula φ such that $q\lceil_{\mathcal{D}} = \big(\mathsf{d\text{-}mng}^{-}(\varphi)\big)\lceil_{\mathcal{D}}$.*

ii) A quasi-query q is finite well-typed generic *iff it is locally a Codd query, i.e. for each finite set $\mathcal{D}$ of db's there is a Codd formula φ such that $q\lceil_{\mathcal{D}} = \big(\mathsf{d\text{-}mng}^{-}(\varphi)\big)\lceil_{\mathcal{D}}$.*

Proof.
($\leftarrow$) of *i)*: From Claim 3.3 we know that a locally first-order quasi-query is generic, and it must be well-typed, because if $\mathfrak{D}_1$, $\mathfrak{D}_2$ are db's then we choose a φ such that $q(\mathfrak{D}_i) = \mathsf{d\text{-}mng}^{-}_{\mathfrak{D}_i}(\varphi)$, and we may see that for $i = 1, 2$ $\mathsf{d\text{-}mng}^{-}_{\mathfrak{D}_i}(\varphi)$ is a relation of sort of $\mathsf{frv}(\varphi)$.

($\rightarrow$) of *ii)*: Let $\mathfrak{D}_1, \ldots, \mathfrak{D}_n$ be db's. Then we can make closed formulas $\varphi_1, \ldots, \varphi_n$ similar to $(*)$ in Section 2, which satisfy the following condition: for any db $\mathfrak{D}$, $\mathfrak{D} \models \varphi_i$ iff $\mathfrak{D} \cong \mathfrak{D}_i$, i.e., φ_i is a description of $\mathfrak{D}_i$.

Using genericity and referring to Claim 3.3, we can also find formulas $\psi_1, \ldots, \psi_n$ such that $q(\mathfrak{D}_i) = \mathsf{d\text{-}mng}^{-}_{\mathfrak{D}_i}(\psi_i)$; because of well-typedness, for all i $\mathsf{frv}(\psi_i)$ is the same set X. From these follows that the the query defined by the formula $\varphi \approx (\varphi_1 \to \psi_1) \wedge \ldots \wedge (\varphi_n \to \psi_n)$ will give the same result as q on $\mathfrak{D}_1, \ldots, \mathfrak{D}_n$.

ii): Here we prove only the non-trivial direction, $(\to)$. In a situation like above, we transform the φ of the preceeding paragraph to a Codd formula in the usual way, based on the observation that because of finiteness, for all i $q(\mathfrak{D}_i) \subseteq \mathsf{adom}(\mathfrak{D}_i)$. ■

Here is an example of a finite well-typed generic quasi-query, which is not recursive: we take an enumeration $t_1, t_2, \ldots$ of Turing machines. Then for a db $\mathfrak{D}$ let

$$q(\mathfrak{D}) = \begin{cases} \{\langle\,\rangle\} & \text{if the Turing machine } t_{|\,\mathsf{adom}(\mathfrak{D})|} \text{ halts on the empty input,} \\ \varnothing & \text{otherwise.} \end{cases}$$

Now we discussed the notion of computable query, next question is: how to handle them?

As we can handle Codd queries via Codd formulas, we could also handle other queries via other query languages. The notion of "query language" was used a few times up to this point, in an intuitive sense. Neither here will we give an absolutely formal definition for "query language"; anyway, we give a by-and-large description of our idea in mind when using this expression, and we also settle some agreements (cf. [ChaHa]).

A ***query language*** consist of a ***syntactic part***, and of a ***semantic part***.

The syntactic part F looks like Fml, or CFml: from some basic symbols we build up strings called formulas, or expressions, with an iterated use of connectives; these connectives may be partial, e.g., disjunction at CFml.

The semantic part is usually given by a ***meaning function*** m, which is a partial mapping $F \times \{\text{databases}\} \longrightarrow \{\text{relations over } U_0\}$. Its value on a given $\varphi \in F$ and a db $\mathfrak{D}$ is denoted by $m_{\mathfrak{D}}(\varphi)$; the unary mappings derived from m by fixing one of its positions are denoted by $m_{\mathfrak{D}}$ and $m(\varphi)$.

We demand that for each $\varphi \in F$ $m(\varphi)$ ("*the query defined by* φ") is a *partial computable query* (hence we remain consistent with our conventions on the usage of "query").

The semantics of naturally arising query languages usually admit a natural extension to the case when we omit the universal domain assumption.

Sometimes we allow "partializing" languages: if some query is definable by a given language, then its restriction to some set of db's is also considered to be definable in that language (provided the restriction is also partial recursive).

Two languages are said to be ***equivalent*** if they define the same class of queries (here we do not allow partializing).

The connection between queries and query languages is usually realized by means of various notions of ***completeness***, ***expressiveness***, and ***boundedness***. A broad class of these notions can be fitted in the scheme of "completeness / expressiveness / boundedness w.r.t. a given class $\mathcal{Q}$ of queries": a query language L is

- ***expressible*** w.r.t. $\mathcal{Q}$, if every query of $\mathcal{Q}$ can be defined by some expression of L;
- ***bounded*** w.r.t. $\mathcal{Q}$, if each expression of L defines a query of $\mathcal{Q}$;
- ***complete*** w.r.t. $\mathcal{Q}$, if expressive and bounded w.r.t. $\mathcal{Q}$.

(we can also say $\mathcal{Q}$-expressive, etc.)

For example, *relational completeness* (cf. Remark 1.18) means completeness w.r.t. Codd queries (of course, Codd queries are always computable, since they can be evaluated using only the finite active domain, so they do not violate our conventions on usage of "query"). Here the usual problem associated to notion of completeness – finding a language which is complete in the given sense – becomes trivial, since the class of queries, according to which we define completeness, is directly given as the class of definable queries of a specific language.

Completeness w.r.t. the class of those queries which are defined exactly on one isomorphism class of databases is called *BP-completeness*. BP-completeness of a language means an ability of expressing those finite relations which are definable by generic means, i.e. automorphism-invariant finite relations. Claim 3.4 and Corollary 3.5 can be summarized as "CFml is BP-complete". This is the first place where we find a *justification of* CFml; up to this point we had no argument against the idea that CFml is an ad hoc choice.

Similarly, Claims 3.3, 3.6 give some kind of justification of first-order logic as a language for defining quasi-queries; not as if for a moment we would have thought that Fml gives an ad hoc way of (quasi-)query defining: Fml is a too much natural language to be in need of justification. Since at BP-completeness the domains of the queries to be defined by a language is quite narrow (as narrow as possible), different BP-complete languages can be very far from being equivalent: e.g., any relational expressive language is BP-complete.

A very natural demand is the demand of finding a language which is *complete w.r.t. the class of all (partial) queries.* E.g., the languages QL, QL_{hs} (see [ChaHa], Definition 4.7, [HiHa]) have this property. This is usually referred to simply as *completeness*, or if we want to emphasize, what kind of completeness, we may say *computational completeness.*

However, there are quite naturally arising classes of queries which does not seem to admit a "reasonable" complete language: e.g., classes of queries of a given complexity. Hence "completeness of a language w.r.t. a complexity class" is defined in way that does not fit our scheme.

How does the map of query languages look like?

CFml (or we can say, the relational complete languages are) is the very basic query language(s). There are some reducts of it in use, like conjuctive queries, and a hierarchy of languages is built on CFml (enrichening it e.g. with iteration features like "fixpoint" or "while"), up to computable complete language. The levels of this hierarchy are labeled with expressibily and complexity classes. The higher expressible the higher complex a query language is. Here we can see the use of low expressible languages: they also have low complexity, thus if for some specfic aim they are suitable, then it is worth to use them.

(In the following we omit the universal domain assumption.)

Thus CFml has a central role; and we feel that using CFml is a very natural way of defining queries. Even we feel that defining first-order queries by Fml is also a very natural activity, but we know that there is one condition of being a computable query which is not satisfied by first-order queries generally: *finiteness.*

Traditionally the non-finite character of first-order formulas is handled via domain independence (see Definition 1.13), or *safety.* "Safety of a formula" has various formal definitions, by-and-large it means domain independence — some authors use safety directly as a synonim for domain independence, some authors call a formula safe if it fulfils some easy-to-check syntactic criteria, and to each domain independent formula they can find a safe one such that these two are equivalent as queries (for example, CFml has this property, see Claim 1.14). Usually it is said that domain independent queries are the reasonable first-order queries. Anyway, there are finite first-order queries which are not domain independent, here we recall the example $\exists x\, \neg \mathsf{R}(x)$.

We try to sketch the possible arguments for considering domain independent queries as reasonable first-order queries, rather that finite ones.

i) Since we want to manipulate data *effectively*, the process of evaluation must depend on only finitely many parameters. This is not the case if we have to execute evaluation on the whole infinite universe.

ii) Even if we find a finite way to evaluate domain-dependent queries, the need of handling the co-finite complement of the active domain surely increases complexity.

iii) Set the universal domain assumption. Then the universal domain is considered as an "infinite ocean of possibly stored data", and a given database $\mathfrak{D}$ carries the actually stored data. We may say, the identity of those elements of the universal domain which are out of the active domain of $\mathfrak{D}$ is not included in the information determining $\mathfrak{D}$; it is considered as a meta-information. Thus the information determining $\mathfrak{D}$ is directly finite, it does not use any recursive encoding of the infinite, it is simply a finite collection of finite tables. Hence the evaluation of queries must not depend on the "outer part". (Especially the same result should be obtained even if we assume the emptiness of the outer part, i.e. if we restrict the evaluation to the active domain — this is *active domain semantics*, see Remark 1.18.)

Although each of the three arguments claim that domain independent queries are more preferable than finite firest-order queries in general, only the first casts doubt on the very use of speaking of finite but domain dependent first-order queries: it disputes the computability of these. Anyway, it turns out that *i)* is a misleading naive approach: by Theorem 2.1, if φ is a finite query with n variables, then, if we want to evaluate it, we can throw away the infinite complement of the active domain, and use n pieces of "invented values" symbolizing the outside part, and we still get the same result (it is independent of the choice of the invented values, as the result will be in the active domain, by Claim 1.8).

The answer to *ii)* is very simple: we may not know until we try — we will return to this issue later on, cf. Theorem 4.6. Moreover, there are two possible cases:

- Finite first-order queries are more expressible than domain independent ones. In this case we may not think of higher complexity as a handicap.

- Finite first-order queries are as expressible as domain independent ones. In this the difference between the two notions becomes syntactic: they present two different ways of describing the same set of computable queries, and it makes no sense to speak of the domain independence of a first-order definable computable query. However, we still may think that finding a domain-independent definition of a computable query enables us to compute it more effectively.

We debate *iii)* from the viewpoint of computability: even if domain independence is a semantic notion, it is meaningless in the context of *semantics of computable queries* — "restriction to a subdomain" is undefined for computable queries. Hence the dependence on the outer values is illusive: one gives you a precisely defined query and a precisely defined database, it is up to you to find the way and where of evaluating, anyway the output is well defined.

Here we can raise another objection: we may use the universal domain assumption in our mathematical model, but this is an abstraction, in reality there is no such a well-defined universal domain, hence in the praxis domain dependent queries may become indefinite.

The answer of the theorist of all ages: "You gave me a mathematical model, now let me use it consequently."

One possible answer of some practician: "So you are interested in practical issues? Come where the going gets tough, and you will find that in praxis there are no implicit dependencies on the outer world: if you ask a query the answer of which depends on the outer world, then your query was not finite." (Here we used the word "query" in an intuitive sense).

We will return to this problem in the beginning of the next section.

We give one more reson for dealing with finite queries: there were introduced a bunch of alternative semantics for evaluating first-order queries, like the aforementioned active domain semantics ([HuSu] presents a nice survey of these). Any of these semantics is easy to simulate with usual, so to say, "unrestricted" semantic and a slight (i.e., yielding an $O(1)$ factor of growth in size) modification on the formula to evaluate. Thus, we can say, unrestricted semantics is the ultimate first-order semantics. Hence the task of characterizing the class of computable queries induced by this semantics is given to us.

So we made up our mind to ask: ***which first-order queries are finite?*** It immediately turns out that we cannot hope a direct answer, that is, we cannot hope a theorem saying: "a first-order formula defines a finite query iff it looks like this and that", because we have the following theorem:

Theorem 3.7 *It is undecidable whether a first-order formula defines a finite query; the same holds with domain independence.*

Proof. We cite a well-known theorem of finite model theory, ***Trahtenbrot's theorem*** (see e.g. [EbFl]): *it is not decicable whether a first-order formula is satisfiable in some finite relational structure.*

Assume we have a decision procedure for finiteness or domain independence. To each formula φ we will show a formula ψ which is finite iff domain independent iff φ is unsatisfiable in finite structures. Hence with the help our decision procedure we will be able to decide finite satisfiability, but this condtradicts Tahtenbrot's theorem.

How to construct ψ? We may assume that φ is closed. Let R be a new unary relation symbol. Let $\psi \approx \varphi\lceil_{\mathsf{R}(x)} \wedge \exists x\,\mathbf{T} \wedge \neg\mathsf{R}(x)$ (the conjuct $\exists x\,\mathbf{T}$ is to keep those databases under control, where R has empty interpretation: $\mathsf{d\text{-}mng}^-_{\circ}(\exists x\,\mathbf{T}) = \varnothing$).

If we can find a finite structure $\mathfrak{M}$ such that $\mathfrak{M} \models \varphi$, then widening $\mathfrak{M}$ with infinitely many new elements and declaring the universe of $\mathfrak{M}$ as the interpretation of R, then we get a database $\mathfrak{D}$ such that $\mathfrak{D} \models \varphi\lceil_{\mathsf{R}(x)}$, hence in $\mathfrak{D}$ ψ is equivalent with $\neg\mathsf{R}(x)$, which spoils its finiteness and domain independence.

If no finite structure satisfies φ, then no database satisfies $\varphi\lceil_{\mathsf{R}(x)} \land \exists x\, \mathbf{T}$, hence ψ, as a query, is non-typed equivalent with $\mathbf{F}$, thus it is finite and domain-independent.

■

The key to finite queries will be completeness: *find a query language which is complete w.r.t. the class of total first-order definable computable queries.* We will call this kind of completeness ***finite first-order completeness***.

Theorem 3.8 (Finite first-order completeness of Codd formulas)
The language of Codd formulas is finite first-order complete.

Proof. We have to prove only finite first-order expressiveness.

Let φ be a first-order formula which defines a finite query. Let $\delta \approx \mathsf{adom}_\varphi$, $D_{\mathfrak{M}} = \mathsf{adom}_{\mathfrak{M}}(\varphi)$.

According to Theorem 2.7, there is a formula φ' such that $\mathsf{d\text{-}mng}^-_{\mathfrak{M}}(\varphi) \cap (D_{\mathfrak{M}})^{\mathsf{frv}(\varphi)} = \mathsf{d\text{-}mng}^-_{\mathfrak{M}_\delta}(\varphi')$, where $\mathfrak{M}$ can be an element of $\mathcal{K}_N(\varphi)$ for some $N \geqslant \omega$, especially $\mathfrak{M}$ can be any database.

On the other hand, since φ is finite, by Claim 1.8, for each database $\mathfrak{D}$, $\mathsf{d\text{-}mng}^-_{\mathfrak{D}}(\varphi) \subseteq (D_{\mathfrak{D}})^{\mathsf{frv}(\varphi)}$.

Hence we have that for each database $\mathfrak{D}$

$$\mathsf{d\text{-}mng}^-_{\mathfrak{D}}(\varphi) = \mathsf{d\text{-}mng}^-_{\mathfrak{D}_\delta}(\varphi') = \mathsf{d\text{-}mng}^-_{\mathfrak{D}}\left(\varphi'\lceil_\delta\right),$$

and $\varphi'\lceil_\delta$ is a Codd formula by definition. ■

This is the ultimate justificaton of Codd formulas. As follows from the construction, the transscription to a Codd formula produces an at most $O(2^n)$-times growth of the transscribed formula.

4 Overriding finiteness

Recall our four criteria which should be met by a quasi-query to be considered a computable query: recursivity, genericity, well-typedness and finiteness.

Recursivity is an obvious criterion for being a computable query. Genericity is also a reasonable criterion: it formalizes the intuition that the output of the query must not depend on anything else than the inner structure of data. Well-typedness is a kind of consistency criterion — consistency with *practice* (consider any usual example, like an employee-salary database: here we work with these two fixed attributes and any reasonable query has a well determined attribute set – there is no use of answers like 112 000 Ft for the question "Who is paid the same salary as Mr. Kovács?"), and consistency with established query languages, which are typically typed, like first-order based query languages. On the contrary, finiteness – which is also required because of preactical reasons – means a strong restriction for established query languages. Should practice bear a high relevance and importance, one feels incomfortable that s/he cannot use the fully-fledged first-order apparatus and cannot ask queries in a bibligraphic database like "Which novels are not authored by Jókai?" — this query, called "the Jókai query", will serve as our basic example (here we again use the word "query" in an intuitive way).

In this section we will make an attempt to extend the relational database model in a way that it would be able to comprehend and answer any first-order queries. Anyway, we will not ignore practical issues: first of all, we will briefly discuss how such an extension confronts with praxis. After we get over it, we will find out what properties should a reasonable extension have, and then we show that the most obvious way of extending yields a model which has the desired properties.

Finiteness is assumed because it offers a very direct way of encoding of relation and database instances: storing them in tables. We could say all real databases operate with tables so there is no use of stretch this point, but on the one hand, as time goes by, there may occur applications of database theory which do not fit into the original frames, on the other hand, what about queries like the Jókai query? — maybe, even if it is possible to store the answer of such queries in tables, this is not the most practical way of storage.

Here occurs a problem, which was once mentioned when we discussed domain dependence.

Domain independent queries are self-contained in the sense that their answer does depend only on those objects which are listed in the tables of the input instances (i.e., the active domain of the input instances): if a bibliographic database contains the list of all Jókai authored writings, as well as the list of all Hungarian novels written during the 19th century, we get the list of Jókai-authored novels by intersecting these two, no matters concretely which bibliographic database do we use. (In this example the surrounding bibliographic database played the role of the universe, and the tables Jókai authored..., Hungarian novels... were supposed to be part of some significantly smaller db, e.g., the would-be-bibliography of our should-be-written literature seminar paper.)

Domain dependent finite queries still depend only on the size of the complement of the active domain (w.r.t. the actual universe), and only up to a very limited point (see Theorem 2.1 or Theorems 2.7, 3.8), which is insignificant compared to the size of the universe, as it is suggested by our familiar abstraction of considering it being infinite.

However, we can find that the answer for the Jókai query very intensively depends on the actually used bibliographic database: if we have the absolute information of "novels authored by Jókai" – which must be the same in every entire bibliographic database, or we can say, which always encompanies us in our pocket as a part of our would-be-bibliography –, then we can reconstruct the whole bibliography database from these two.

We could say, it is even worse, if mean the question of the Jókai query, and we are curious to know of any novel which has been and will be written in whole wide world, by anybody who is not Jókai. Then it is impossible to handle the records of which the answer of our query consists. Hence we cannot identify and manage this answer as a list of records; although it is very easily identified by its complement, the reasonably lenghty list of novels authored byJókai.

So we can say, the worse is the better, and saying goodbye to the "answer of a query is a list of records" concept, we can find a new kind of absoluteness of the answer: *the answer is identified with a kind of abstract representant of the logical condition, by which it is defined.* In the idea of "representant" it is included in the idea of "representant" that the same representant is assciated with equivalent queries. Immediately the following questions spring up: in what sense is "equivalence of queries" meant, what is a "representant" sintactically, and how to decide equivalence of queries, how to compute the representants? The notion of equivalence is quite clear: according to our orginal idea, the universe is considered to be infinite, which condition is strict enough to ensure us: if two queries give different answers on a given istance at some choice of the universe, then they give different answers on that instance at any choice of the universe. THe other questions can be left to the technicians, we have set the paradigm.

And at any particular choice of the universe we may set the implementation of the whole abstract model. The implementation is universe-dependent, the model is not.

There is one more way to utilize these concepts: if the active domain of our database is relatively large, then, if we execute queries in the above described sense, then the answer may serve as some compressed version of the "real answer" (answer given by a table), which is easier to store and is easier to query with. Since we have the elements of the active domain listed in a table, decompression is possible.

The distinctions made up to this point will not appear explicitely in the structure of our formal model, since the formal model avoids these ambiguities, difficulties simply by including the universe as a part of the information determining a database, ignoring whether it is possible in praxis, when an infinite, or even worse, "effectively infinite" (finite but too big to handle within a reasonable amount of time) universe occurs. This was the case even with classical database theory — the reasons for preferring domain independent queries to dependent ones lie out of the scope of the theory.

Now we enumerate the properties which a reasonable extension should have. The idea is that every construction we realized in the original model, must be realizable in the extended one by a natural extension, and in the same way, with the same features.

***a*)** The basic elements should be the same: some kind of relational structures, on which we execute queries, which are usually defined by query languages. The relation instances in the input must be "of the same kind" as output relations, i.e. queries must be composable.

***b*)** Each considered relation instance must have a finite encoding which enables us decide in linear time whether a given tuple of the universe is included in the instance.

***c*)** We must be able to decide in linear time whether a given relation instance is an old kind of instance, or an extended one. Especially computable queries in the old sense will be computable queries in the extended sense, if we declare them undefined on the extended instances.

***d*)** Any first-order formula must define a total computable query.

Complexity of evaluating computable first-order queries must not be affected by this extension.

***e*)** Some computationally complete query language in the old sense must extend naturally to the new instances, yielding a computationally complete query language in the extended sense.

Our extension will be the smallest possible which has the chance to satisfy ***d***):

Definition 4.1 If U is a set, we will say that a relation on U is ***finitely generated***, if it is first-order expressible from some finite relations. With other words, a relation is finitely generated, if it is in $[\![\mathrm{RE}D]\!]_n$ for some n and for some finite subset D of U (cf. Claim 2.2) (we could be more precise by writing $[\![\mathrm{RE}D]\!]_X$, where X is a finite subset of Att).

An ***extended database*** is a relational structure such that its universe is infinite, and each relation symbol is interpreted to a finitely generated relation. □

We will show that this extension fulfils each of the criteria a) – e).

Our extension fulfils **Criterion *a*)** by its very definition.

Criterion *b*). Now we are to find out by which finite means should we refer to finitely generated relations on some infinite set U.

If fix an n, and a finite $D \subseteq U$, then the following method seems to be the most obvious:

Any element of $[\![\mathrm{RE}D]\!] = [\![\mathrm{RE}D]\!]_n$ is a finite union of some atoms $A_{f\vartheta}$, as we discussed it in Theorem 2.1. Hence all we have to do is list the involved $A_{f\vartheta}$'s in a table. Technically this is carried out in the following way: we introduce an infite sequence of new symbols, $\infty_1, \dots, \infty_i, \dots$, called the *invented values*. Then our table will have n columns, and at each entry there can be either an element of D or an invented value ∞_i. Say the tuple $\langle 1 : d_1,\ 2 : \infty_1,\ 3 : d_2,\ 4 : \infty_1,\ 5 : \infty_2\rangle$ is a row of our table. Then let f be $\langle 1 : d_1, 3 : d_2\rangle$, and ϑ be the equivalence on $\{2, 4, 5\}$, by which, in addition to the identic pairs, 2 and 4 are declared to be equivalent. The atom denominated by the tuple will be $A_{f\vartheta} = A^D_{f\vartheta}$. This way several tuples can name the same atom, and tables may become redundant, if they have distinct rows naming the same atom, but it makes no trouble, because it is easy to decide whether two tuples name the same atom, hence it is also easy to eliminate redundance of tables. Or if we feel like doing so, we may canonize a name for each atom by putting an order on Att and the invented values.

This is the "tuples with invented values", or "tuples with placeholders" concept, which was used in [AGSS] and in [AHV, Theorem 5.6.1] to describe answers of first order queries in general.

Now, if we do not have a fixed D, and we intend to specify finitely generated relations on U, we do not introduce invented values, we let elements of U to behave like invented values sometimes.

We use At_D for $\{A^D_{f\vartheta} : f, \vartheta\}$. For each finite $D \subseteq U$ we associate a mapping $A_D : U^n \longrightarrow At_D$ such that for $t \in U^n$ $A_D(t)$ is the the atom $A^D_{f\vartheta}$ which we get by considering the coordinates of t out of D as invented values. More precisely, declaring $X = t^{-1}(D)$ and $X \dot\cup Y = n$, $A_D(t) = A_{t\restriction_X, \mathsf{Ker}(t\restriction_Y)}$.

Then the finitely generated n-ary relations on U will be encoded by pairs $\langle D, R\rangle$, where D is a finite subset of U, and R is a finite n-ary relation on U. Of course, a pair $\langle D, R\rangle$ encodes the relation $A_D(R) = \{A_D(t) : t \in R\}$. We may say, various D's define various alternative semantics for tables.

Here we face a more serious ambiguity than the interchangeability of invented values: a finitely generated relation can be encoded via different D's — e.g., $A_D(R)$ can be given in the form $A_{D'}(R')$ for any $D' \supseteq D$.

Now we discuss how this ambiguity can be dealt with.

The equation "$Q = A_D(R)$" can be read as "R represents Q w.r.t. D", or "R is a representation of Q w.r.t. D". A relation is *D-representable*, if it has a representation w.r.t. D. We say a subset X of some set H *respects* a given equivalence Θ of H, if X is the union of some blocks of Θ. If Θ, Θ' are equivalences, then $\Theta \vee \Theta'$ denotes the smallest equivalence containing both of them. The reader may see that a set respects $\Theta \vee \Theta'$ iff it respects both of them.

For $D \subseteq U$ let S_D be the group of those permutations of U, which keep D elementwise fixed (the *stabilizator group of D*). Let Θ_D be that equivalence of U^n, the blocks of which are exactly the orbits of $S_D^{[n]}$ (we remind the reader that the notation $G^{[n]}$ was defined in the proof of Claim 2.2).

We have discussed in the proof of Claim 2.2 that the atoms $A^D_{f\vartheta}$ are exactly the orbits of $S_D^{[n]}$. Hence if D is a finite subset of U, then an n-ary relation is D-representable iff it respects Θ_D; besides, for any $R \subseteq U^n$, $A_D(R)$ is the union of those $G^{[n]}$-orbits (Θ_D-blocks) which are intersected by R.

It is straightforward that $\Theta_D \vee \Theta_{D'} = \Theta_{D\cap D'}$. Thus a relation is D-representable and D'-representable iff it is $D \cap D'$-representable.

Hence if Q is a finitely generated relation, then there is a smallest D_0 such that Q is D_0-representable: if Q is, say, D-representable for some finite D, then we get D_0 as $\bigcap\{D' \subseteq D : Q \text{ is } D'\text{-representable}\} = \bigcap\{D \setminus \{u\} : u \in D,\ Q \text{ is } D \setminus \{u\}\text{-representable}\}$. We will call this D_0 the ***active domain of*** Q, denoted by $\mathsf{adom}(Q)$.

If a relation Q is D-representable and $D' \supseteq D$, then, as we mentioned, it is also D'-representable, but $\Theta_{D'} \subseteq \Theta_D$, thus the former has smaller and more blocks than the latter, the number of $\Theta_{D'}$-blocks included in Q bigger than the number of Θ_D-blocks included in Q, which means that a representation w.r.t. D' must be bigger than a (non-redundant) representation w.r.t. D.

Summarizing these, *the optimal representations are the representations w.r.t. active domain.*

Given some $Q = A_D(R)$, how to find the active domain, and a representation w.r.t. the active domain?

$D-u$ will abbreviate $D \smallsetminus \{u\}$. By the above given form of $\mathsf{adom}(Q)$ it is sufficient to check for all $u \in D$ whether R is $(D-u)$-representable. And the same R will do as a representant w.r.t. the active domain — although it may be a redundant representant.

How to check $(D-u)$-representability? We have to check whether for $t \in R$ $t/\Theta_{D-u} \subseteq Q = A_D(R)$. This can be done by identifying the Θ_D-blocks included in t/Θ_{D-u}, and checking whether have they a representant in R (w.r.t. D). The details of this is left to the reader. This yields an algorithm for finding the active domain which can be executed in $O(n^3\,|R|^2\,|D|)$ time (where the time of deciding the equality of two elements of U was taken for time unit).

By putting an order on Att, and by replacing the occurrences of elements of $U \smallsetminus \mathsf{adom}$ in R with ordered invented values, we may get a representation (w.r.t. $D = \mathsf{adom}$), which is unique up to permutation of rows and permutation of elements of D.

However, usually we consider any non-redundant adom-representation as satisfactoty.

When we define constructions on finitely generated relations, usually we define just *some* kind of representation of the thing to be constructed, without taking care about the optimality of the defined representation. By the above, optimizing the representation is possible and a mechanic task, thus we do not need to care about it.

Criterion *c*). Deciding the finiteness of a finitely generated relation given by its representation (D, R) is very easy: the only thing we are to do is to check whether every element, which occurs in R, also occurs in D.

Criterion *d*). Extending the notions of quasi-query, and its possible properties except for finiteness to our model is straightforward. Here we will consider a quasi-query as a ***computable query***, if it is *recursive, generic, well-typed.*

Of course, in our extension first-order formulas define well-typed generic quasi-queries. We have to see computability.

The implementation of (at most) unary connectives is straightforward: if we have a finitely generated relation Q with active domain $D = \mathsf{adom}\, Q$, given by a representation (D, R), then $Q \in [\![\mathrm{RE}D]\!]$, which is a finite Cs_n with a well-known structure as it is described in Theorem 2.1, and we simply compute the effect of the connective within $[\![\mathrm{RE}D]\!]$. Note that negation (complementation) behaves a quite interesting way: it is very easy to compute in the sense that time of computation is linear in the size of the output, but this computation may take a very long time, i.e. outputs can be quite big: an atom's complement is of size $|\,[\![\mathrm{RE}D]\!]_n\,| - 1$, which is exponential in n: $(|D|+1)^n \leqslant |\,[\![\mathrm{RE}D]\!]_n\,| \leqslant (|D|+n)^n$.

The case of binary connectives is a bit more difficult: here we are given two relations, which may have different active domains. Still we can execute the computation in $[\![\mathrm{RE}D]\!]$ where D is the union of their active domains, but the redundancy appearing with this change of domain is so large that this computation will be far not optimal.

As an example, we will show how to proceed in case of conjunction ($\wedge$). Its algebraic counterpart (by $\mathsf{d\text{-}mng}$) is the join operator $\bowtie$; however, for the sake of simplicity we will proceed with relations of the same arity, in which case join is simply intersection (this case can be generalized easily).

There is a primitive but not so bad algorithm for computing intersection of some relations R_1, R_2 in usual, finite databases: we take one element of the first one, and compare it with each element of the other one; if we find it equal to some, we keep this element, otherwise we throw it away. This means $|R_1|\,|R_2|$ comparisons of tuples, i.e. $n\,|R_1|\,|R_2|$ comparisons of elements.

This could be applied in the finitely generated case as well, if we could execute comparisons of tuples also in this case. What to mean by that?

Say, we intend to intersect $Q_1 = (D_1, R_1)$ and $Q_2 = (D_2, R_2)$. We claim that

Lemma 4.2 $\Theta_{D_1} \cap \Theta_{D_2} = \Theta_{D_1 \cup D_2}$.

Proof. The $\supseteq$ direction is obvious. On the other hand, if $t, t' \in U^n$, then there is a permutation σ such that $\sigma^{[n]}(t) = t'$ iff the relation $b = \{\langle t_i, t'_i\rangle : i = 1, \ldots, n\}$ is a bijection. If $\langle t, t'\rangle \in \Theta_{D_1} \cap \Theta_{D_2}$, then it surely holds, and any σ permutation of U for which $\sigma^{[n]}(t) = t'$, is the extension of this bijection. So because $\langle t, t'\rangle \in \Theta_{D_1}$, b fixes the points of D_1, and similarly, it fixes the points of D_2. Extending arbitrarily b to U such that it still fixes points of D_1 and D_2, we have verified $\langle t, t'\rangle \in \Theta_{D_1 \cup D_2}$. ■

So to get a representation of $Q_1 \cap Q_2$ w.r.t. $D_1 \cup D_2$, we only have to find an element of the set $t_1/D_1 \cap t_2/D_2$, if is there any (where t_i runs over R_i, $i = 1, 2$). It is easy to see that $t_1/D_1 \cap t_2/D_2 \neq 0$ iff $\mathsf{Ker}(t_1) = \mathsf{Ker}(t_2)$, and there is no $j < n$ such that $(t_1)_j \in D_1$, $(t_2)_j \in D_2$, and $(t_1)_j \neq (t_2)_j$. If this condition holds, then it is easy to find an element in the intersection.

The hardest part of this "tuple comparison" is comparising the kernels, anyway, it involves no more than n^2 comparisons between elements of U. Hence in $n^2\,|R_1|\,|R_2|$ time we found a representation of the intersection of size at most $n^2\,|R_1|\,|R_2|$.

Using this "tuple comparison" technique, more subtle algorithms for intersection can also be imported from the finite case.

Now we turn to criteria ***d***), ***e***). Here we intend to adapt classical database-theoretic results for our extension. Hence we think it is not a task of us to re-prove them, we will just try to show the crucial points of the constructions, and we will motivate that they are preseved by the extension. *We set the universal domain assumption*; especially the type we work with is finite.

Criterion *d*). Informally we referred to complexity of queries earlier, but we have not defined it yet.

There are several versions of query complexity.

E.g., [Grohe] mentions *combined complexity* "...under which the complexity of the evaluation problem is measured in terms of the size of the database plus the length of the query". Because it turned out to be a too rough tool for scaling complexity of query language, relatively recently a much more subtle notion of query language complexity has been introduced: the so-called *parametric complexity*, which is to measure the complexity of questions like "is there a k-clique in the graph?" — here n is the parameter, and problems where the dependence of the parameter is like $n^{f(k)}$ are considered to be hard, and where the dependence of the parameter is like $g(k)n^c$ are considered to be tractable. Like in classical complexity theory NP-COMPLETE is considered to be hard (though we do not know whether this level of complexity implies being exponential), in parametric complexity theory there is hierarchy of hard problems, the so-called AW-hierarchy (but we do not know whether this level of complexity implies being of the first kind of parametric form).

Independently of their roughness or subtleness, *these notions of query-complexity are irrelevant to our aims*: they show that first-order based query languages are very hard — combined complexity classifies CFml as PSPACE-COMPLETE (cf. [Vardi]), and parametric complexity classifies CFml as $W[P]$-HARD (cf. [PaYa], [Grohe]), which latter class belongs to the AW-hierarchy.

Hence these notions can serve only as an *upper bound* of complexity, while we intend to prove that a *lower bound* of complexity is preserved by our extension, since we want to show that our model is of low complexity.

There is a commonly accepted notion of query language complexity: ***data complexity***, which shows a low-complexty side of first-order based query languages. It poses the following question: *if we have a fixed query of the language, then for an input db $\mathfrak{D}$ and an input tuple t, how hard is it to decide (in terms of $|\mathfrak{D}|$) whether t is included in the output of the query evaluated in $\mathfrak{D}$?* This question reflects our usual experience that the size of query is as small that it can be considered as a constant.

Definition 4.3 AC_0 is the class of those decision problems, which can be answered by a uniform family of Boolean circuits (i.e., for each input size m we have a Boolean circuit with m input nodes, and this family of circuits is recursively definiable) such that the depth of the circuits is constant but the arity of the AND and OR nodes is arbitrary, and the number of nodes is polynomial. Intuitvely, AC_0 is the class of those problems, which can be aswered in constant time with a paralell computation using polynomially many processors. □

Theorem 4.4 ([Papa]) $\mathrm{AC}_0 \subseteq \mathrm{LOGSPACE} \subseteq \mathrm{P}$

Theorem 4.5 ([AHV]) *Data complexity of* CFml *is in* AC_0.

Theorem 4.6 *Data complexity of* Fml *in the extended relational model is in* AC_0.

Proof (sketchy). The following was the situation in the classical finite case:

"So we have a fix formula. Because CFml is domain independent, we may execute our computation in adom^n, where n is the number of free variables of the formula. This is how to construct the circuit with k input nodes:

We can take a set D_k which is big enough to include the possible adom's of inputs of size k. Then we organize the nodes so as to they execute the process of evaluation. First of all, there will be nodes which correspond to points of D_k^n, and they will send "1" on their output wire iff they correspond to a point included in the input. Then we execute the evaluation point-by-point — the only step where we seem to be forced to consider more than one point simultaneously, is existential quantification, but since we evaluate over a concretely given finite set, it can be taken as a big disjunction. The depth will be constant, approximately the depth of our formula (the elementary steps of evaluation are realized by small blocks of nodes, the depth of these blocks multiplies the depth of the formula, but these are all constant quantities). The number of nodes will be commensurable to the number of points, $|D_k|^n$."

What is different when we are querying on finitely generated relations?

Surely not the way of evaluation. Only the scene of the evaluation process is to be chosen more carefully; again, since the type is finite, we can find a set D_k which is big enough to be able to include the possible adom's of inputs of size k. Then we add n pieces of invented values – now this counts in as a constant growth in the size of the universe – from this point everything may happen as above. ■

Criterion *e*). Our candidate of a complete language will be QL_{hs}, which was proved to be computationally complete in the usual model, and for the so-called *highly symmetric recursive databases* (see [ChaHa], [HiHa]). It is defined as follows:

Definition 4.7 ***The language*** QL_{hs} has a symbol E referring to equality, and an infinite colletion of relation variables, $y_1, y_2, \ldots$. The *terms* of QL_{hs} are given recursively:

- E, the relation symbols of the type, and y_i are terms.
- if e, e' are terms, then $e \cap e'$, $\neg e$, $e\uparrow$, $e\downarrow$, $e\sim$ are terms.

We also define the *programs* of QL_{hs}:

- $y_i \leftarrow e$ is a program for all i.
- if P, P' are programs, then $(P; P')$, *while* $y_i = \varnothing$ *do* P and *while* $|y_i| = 1$ *do* P are also programs.

Besides, we assume that there is given an enumeration on Att:
$\mathsf{Att} = \{x_1, x_2, \ldots, x_n, \ldots\}$.

Now we give the semantics in some database $\mathfrak{D}$: relation symbols are interpreted in the same way as in $\mathfrak{D}$, E has to be interpreted to the real equailty relation, $\cap$ act as intersection, $\neg$ as complementation (in [ChaHa] it was taken w.r.t. the active domain, in the finitely generated case w.r.t. the universe).

$\downarrow$ is projecting out the first coordinate, i.e. if R is a relation of short $X = \{x_{i_1}, \ldots, x_{i_n}\}$ (where $i_1 < i_2 < \ldots < i_n$), then $R\downarrow = \pi_{X - x_1} R$. $R\uparrow = R \times U_0^{\{x_{i_n+1}\}}$. $\sim$ interchanges the last two coordinates.

As [ChaHa] says, programs in QL_{hs} "act in the obvious way..." □

Recall the notations $G[n]$, Aut from Claim 2.2, Corollary 2.6. The following notions are introduced in [HiHa]:

A relation structure $\mathfrak{M}$ is *highly symmetric*, if for all n, $\mathsf{Aut}_{\mathfrak{M}}^{[n]}$ has finitely many orbits. A relational structure is a highly symmetric recursive database if it is a highly symmetric relational structure with an infinite universe, the interperations of the relation symbols are recursive relations, and there is given a special kind of encoding of the interpretations, which we do not give here, the reader may see it in [HiHa]. A relational structure is a finite–co-finite database, if its universe is infinite, and all the interpreted relations are either finite or co-finite, and the interpretations are encoded by pairs consisting of a number $\in \{-1, 1\}$, and a table (if the number is -1, the interpretation is the complement of the table-defined relation).

Theorem 4.8 ([HiHa]) QL_{hs} *is complete for highly symmetric recursive databases and for finite–co-finite databases.* ■

The proof of this theorem can be interpreted to our case, and the proof for finite–co-finite databases is also valid for our finitely generated databases, hence we have

Theorem 4.9 QL_{hs} *is computably complete in extended model.*

Summarizing the results:

Theorem 4.10 *The extension of finitely generated databases fulfils Critera* **a)** *–* **e)***.* ∎

Remark 4.11 (Historical remarks) There has been some attempts to override finiteness. [ImLi] mentioned that using invented values, unions of relations of different sorts becomes storable in tables. Here the invented values were absolute independent of each other, and could symbolize any element of the universe.

In [HiHa] the *recursive* and *highly symmetric recursive databases* were discussed. Recursive databases mean the broadest reasonable extension (from the viewpoint of computabililty): recursive relations are the ones, which can be encoded finitely such that with this encoding we can answer the memebership problem . However, in all other aspects the behaviour of this extension has nothing to do with our ideas: e.g. in the recursive case the admissible relations are not closed w.r.t. $\exists$, we cannot decide whether a database instance consists of finite relations, and in both of the recursive and highly symmetric recursive cases, complexity issues become meaningless (no bound for complexity).

[AGSS] or [AHV, Theorem 5.6.1] found the same way of representating answers of queries as we did. However, they just wanted to interpret the answer of any first-order query, they did not intend to execute the process of consequent extension. □

References

[AGSS] A.K. Ailamazyan, M.M. Gilula, A.P. Stolboushkin, G.F. Shvarts: *Reduction of a relational model with infinite domains to the finite–domain case*, Dokl. Akad. Nauk SSSR 286 (1986), no. 2, 308–311, in Russian; also Soviet Physics Doklady, 31(1) 11-13, 1986 (English translation).

[AHV] S. Abiteboul, R. Hull, V. Vianu: *Foundations of Databases*, Addison–Wesley, 1995.

[ANS a] H. Andréka, I. Németi, I. Sain: *Algebraic Logic*, in: Handbook of Philosophical Logic, to appear.

[Ba] Bancilhon, François: *On the completeness of query languages for relational data bases*, in: Mathematical foundations of computer science (Proc. Seventh Sympos., Zakopane, 1978), pp. 112–123, Lecture Notes in Comput. Sci., 64, Springer, Berlin–New York, 1978.

[ChaHa] A.K. Chandra, D. Harel: *Computable queries for relational data bases*, Journal of Computer and System Sciences, 21(2):156–178, 1980.

[ChaHa2] A.K. Chandra, D. Harel: *Structure and complexity of relational queries.* Journal of Computer and System Sciences 25(1): 99–128, 1982.

[Codd70] E.F. Codd: *A relational model for large shared data banks*, comm. of the ACM, 13(6): 377–387, 1970

[Codd72] E.F. Codd: *Relational completeness of database sublanguages*, in R. Rustin, editor, Courant Computer Symposium 6: Data Base Systems, pp. 33–64.Prentice–Hall, Englewood Cliffs, NJ, 1972

[Cosmadakis] S.S. Cosmadakis: *Database theory and cylindric lattices*, in.: A.K. Chandra (Ed.), Proceedings of the 28th Annual Symposium on Foundations of Computer Science, 411–420, Los Angeles, CA. IEEE Computer Society Press, 1987.

[DM] I. Düntsch, Sz. Mikulás: *Cylindric structures and dependencies in relational databases.* Preprint.

[EbFl] H-D. Ebbinghaus, J. Flum: *Finite model theory*, Second edition. Perspectives in Mathematical Logic. Springer–Verlag, Berlin, 1999. xiv+360 pp.

[Grohe] M. Grohe: *The parametrized complexity of database queries*, available at www.mgi-informatik.rwth-aachen.de/FMT

[HiHa] T. Hirst, D. Harel: *Completeness Results of Recursive Data Bases*, in 12th Annual ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS) (Washington, DC, 1993), 244–252; also Journal of Computer and System Sciences 52 (1996), no. 3, part 2, 522–536.

[HMT1] L. Henkin, J.D. Monk, A. Tarski: *Cylindric Algebras, Part I.*, North–Holland, Amsterdam, 1971, 1985.

[HMT2] L. Henkin, J.D. Monk, A. Tarski: *Cylindric Algebras, Part II.*, North–Holland, Amsterdam, 1985.

[HuSu] R. Hull, J. Su: *Domain independence and the relational calculus*, Acta Informatica 31 (1994), no. 6, 513–524; also available at http://www-db.research.bell-labs.com/user/hull

[ImLi] T. Imieliński, W. Lipski: *The relational model of data and cylindric algebras*, J. Comput. System Sci. 28 (1984), no. 1, 80–102.

[Papa] C.H. Papadimitriou: *Computational complexity*, Addison-Wesley Publishing Company, Reading, MA, 1994. xvi+523 pp.; in Hungarian: Számítási bonyolultság, 1999, Novadat.

[Papa2] C.H. Papadimitriou: *Database metatheory: asking the big queries*, available at http://www.cs.berkeley.edu/˜christos/

[PaYa] C.H. Papadimitriou, M. Yannakakis: *On the complexity of database queries*. Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (Tucson, AZ, 1997). J. Comput. System Sci. 58 (1999), no. 3, 407–427; also available at http://www.cs.berkeley.edu/˜christos/

[Par] J. Paredaens: *On the expressive power of the relational algebra*, Inf. Proc. Letters, 7(2): 107–111, 1978.

[Vardi] M.Y. Vardi: *The Complexity of Relational Query Languages*, In Proc. ACM SIGACT Symp. on the Theory of Computing, pp 137-146, 1982