The expressiveness of Datalog circuits (DAC)

Foto AFRATI [1], Irène GUESSARIAN [2] and Michel de ROUGEMONT [3]

ABSTRACT : We define a new logic query language, called DAC, which is an extension of Datalog. We exhibit queries which are not Datalog expressible but are DAC expressible. We also prove non-expressiveness results for DAC and we infer various strict hierarchies obtained by allowing more rapidly growing functions on the bound parameters.

I Introduction

A Datalog program is an existential inductive definition on a class of finite relational structures which defines a monotone query computable in polynomial time. It has been shown in [3] that there are monotone PTIME-computable queries that are not Datalog expressible, for example the chain query that defines on the class of graphs the existence of a path of length q^2 for some q. We observed that if we fix n, there is a *Datalog* program P_n that defines this query for the class of graphs of size n, and there is a PTIME-computable function g such that for each n, $g(n)$ describes P_n as a string. We will say that such a query is expressible by *DAtalog Circuits* and is in the class *DAC*. The class *DAC* extends *Datalog* by allowing (i) non-constant (with respect to the size of the input structure) size of programs and (ii) an infinite sequence of programs $P_1, \ldots, P_n, \ldots$, each P_n defining the query on structures (or databases) of size n. Namely, program P_n will give the answers whenever the size of the data domain is n. The parameters of the program P_n, now, are functions of n. We describe P_n in terms of the number of rules –we call it height and denote it $h(n)$–, the maximum number of atoms per rule –we call it width and denote it $w(n)$–, and the recursion depth –we call it bound of recursion and denote it $b(n)$. $DAC(w(n), h(n))(b(n))$ is the class of queries that are accepted by such families of programs $\mathcal{P} = \{P_n, n \in \mathbb{N}\}$.

Of particular interest is the class $smDAC(1, poly)(\log n)$ where the programs have constant width, polynomial height and $\log n$ recursion depth and are strongly monotone. This class subsumes (i) both the parallelism and logarithmic depth of the class mAC^1 (monotone AC^1 in [5]) of queries accepted by positive circuits of polynomial size and $\log n$ recursion depth, and (ii) the strong monotonicity of *Datalog*. We first study the case of bounded *DAC*, i.e. the case when the recursion depth is bounded and then the case when the recursion depth is $\log n$. Our main results are:

- $smDAC(1, h(n))(1)$ coincides with Bounded Datalog (Theorem II.1).
- The perfect p-th power query separates the classes $smDAC(1, n^{12})(\log n)$ and $smDAC\big(1, (\log n)^{1-\varepsilon}\big)(\log n)$ for $\varepsilon > 0$ (Theorem III.3).

---

[1] N.T.U.A. 157 73 Zographou, Athens, Greece. Part of this work was done while visiting Université Paris VI and Université Paris XIII. E-mail: afrati@softlab.ece.ntua.gr

[2] L.I.A.F.A. Université Paris VI, 4 Place Jussieu, 75252 Paris Cedex 05, France. Work supported by Esprit Gentzen project no 7232. E-mail: ig@litp.ibp.fr.

[3] L.R.I. Université Paris XI, 91405 Orsay Cedex, France. E-mail: mdr@lri.fr

- Let, for $i > 0$, $n_i = \overbrace{\log\log\ldots\log}^{i \text{ times}} n$; the classes $DAC(1, n_i)(n_i)$ form a strictly decreasing hierarchy (Theorem IV.4).

It is well known [3] that *Datalog* expressible queries are computed by polynomial size monotone circuits. The fragment of DAC that we investigate also expresses queries which can be computed by polynomial size monotone circuits. DAC programs that belong to this fragment are of interest, because they enhance largely the expressive power of *Datalog*, while being at most as expressive as polynomial size monotone circuits. DAC programs allow for trade-offs between the width, the height and the depth of recursion, as illustrated in Example I.1 below:

EXAMPLE I.1 On a graph with a binary edge relation E, the transitive and reflective closure of E, denoted by TC, is

(i) $DAC(1, \log n)(\log n)$. The following program $\mathcal{P} = \{P_n, n \in \mathbb{N}\}$ which has constant width 2 and height $2\lfloor \log n \rfloor + 2$, hence is in $DAC(1, \log n)$, defines TC in $\log n$ parallel computation steps.

$$P_n \quad \left\{ \begin{array}{rl} TC(X,Y) \longleftarrow & TC_{\lfloor \log n \rfloor -1}(X,Z)\, TC_{\lfloor \log n \rfloor -1}(Z,Y) \\ TC(X,Y) \longleftarrow & TC_{\lfloor \log n \rfloor -1}(X,Y) \\ RC(X,Y) \longleftarrow & E(X,Y) \\ RC(X,X) \longleftarrow & \\ TC_1(X,Y) \longleftarrow & RC(X,Z)\, RC(Z,Y) \\ TC_1(X,Y) \longleftarrow & RC(X,Y) \\ & \vdots \\ TC_{\lfloor \log n \rfloor -1}(X,Y) \longleftarrow & TC_{\lfloor \log n \rfloor -2}(X,Z)\, TC_{\lfloor \log n \rfloor -2}(Z,Y) \\ TC_{\lfloor \log n \rfloor -1}(X,Y) \longleftarrow & TC_{\lfloor \log n \rfloor -2}(X,Y) \end{array} \right.$$

(ii) $DAC(n, 1)(1)$. The following program $\mathcal{P}' = \{P'_n, n \in \mathbb{N}\}$ which has width n and height 3, hence is in $DAC(n, 1)$, defines TC in recursion depth 2.

$$P'_n \quad \left\{ \begin{array}{l} TC(X,Y) \longleftarrow RC(X,X_1)\, RC(X_1,X_2)\, \ldots\, RC(X_{n-1},Y) \\ RC(X,Y) \longleftarrow E(X,Y) \\ RC(X,X) \longleftarrow \end{array} \right.$$

(iii) $DAC(\log n/\log\log n, 1)(\log n/\log\log n)$. For any n, let $k(n)$ be the least natural number such that: $k(n)^{k(n)} < n \leq (k(n)+1)^{k(n)+1}$. The following program $\mathcal{P}'' = \{P''_n, n \in \mathbb{N}\}$ which has width $k(n)+1$, two IDB predicates and height 4 defines TC in recursion depth $k(n)+3$. The result follows because $k(n)$ is in $O(\log n/\log\log n)$.

$$P''_n \quad \left\{ \begin{array}{l} TC(X,Y) \longleftarrow RC(X,Y) \\ TC(X,Y) \longleftarrow TC(X,Z_1)\, TC(Z_1,Z_2)\, \ldots\, TC(Z_{k(n)-1},Z_{k(n)})\, TC(Z_{k(n)},Y) \\ RC(X,X) \longleftarrow \\ RC(X,Y) \longleftarrow E(X,Y) \end{array} \right.$$

(iv) Transitive closure is **not** $DAC(\log n, 1)(1)$. Assume $TC \in DAC(\log n, 1)(1)$ and let $\mathcal{P} = \{P_n, n \in \mathbb{N}\} \in DAC(\log n, 1)(1)$ be a DAC program defining TC. Consider now the fringe (see Definition II.2) of a proof tree corresponding to a pair (X, Y) in

$TC_{P_n}(\mathcal{D}_n)$: it consists of a set of atoms $E(Z_i, Z_j)$ which form a path going from X to Y. Because proof trees of programs in $DAC(\log n, 1)(1)$ have depth at most k', and branching width at most $c \log n$, their fringes will correspond to simple paths of length at most $(c \log n)^{k'} < n$; hence P_n cannot capture e.g. the transitive closure on a graph consisting of a single simple path of length n.

(v) Note that TC is a strongly monotone query (see Section II.3), and all our programs are strongly monotone; hence we have the more precise results that TC is in $smDAC(1, \log n)(\log n)$, in $smDAC(n, 1)(1)$ and also in $smDAC(\log n / \log\log n, 1)$ $(\log n / \log\log n)$.

The paper is organized as follows. In the second section, we define DAC programs, we study bounded and monotone DAC programs and state Theorem II.1. In the third section, we state the pumping lemma for DAC programs and Theorem III.3. In the fourth section, we show our hierarchy theorems and Theorem IV.4. The technical proofs are in the full paper, which can be obtained from the authors.

II The definition of DAC

II.1 Datalog

Let D be a finite set; a **database** $\mathcal{D}$ over the domain D is a finite relational structure $\mathcal{D} = (D, R_1, \ldots, R_k)$, the sequence $a = (a_1, \ldots, a_k)$ of arities of the R_is is the type of the database. $R_1, \ldots, R_k$ are called extensional predicates (or EDBs). In the sequel, $\mathcal{D}_n$, or just $\mathcal{D}$, will denote a database having n elements (or entities): n is said to be the **size** of $\mathcal{D}_n$; the size here refers to the number of entities in the database, and **not** to the number of ground facts, or tuples in the database. $\mathbb{D}_n$ will denote the set of databases $\mathcal{D}_n$ having n elements. A **query** Q (of arity p) is a function associating with a database $\mathcal{D}$ a relation $Q(\mathcal{D})$ of arity p over the domain D of $\mathcal{D}$.

Let $T_1, \ldots, T_l$ be new predicate symbols, called intensional predicates (or IDBs). A *Datalog* **program** P is a finite set of function-free Horn clauses, called rules, of the form:

$$Q(X_1, \ldots, X_n) \longleftarrow Q_1(Y_{1,1}, \ldots, Y_{1,n_1}), \ldots, Q_p(Y_{p,1}, \ldots, Y_{p,n_p})$$

where the X_is and the $Y_{i,j}$s are either variables or constants, Q is an IDB, the Q_is are either IDBs or EDBs. For each database $\mathcal{D}$:

- $F(\mathcal{D}) = \{R_j(d_1, \ldots, d_n) \mid d_i \in D,\ \text{for } i = 1, \ldots, n \text{ and } (d_1, \ldots, d_n) \in R_j\}$ is the set of ground atoms which hold in $\mathcal{D}$, also called facts;
- for each *Datalog* program P, the Herbrand basis is the set of ground atoms $B(\mathcal{D}) = \big\{Q(d_1, \ldots, d_n) \mid d_i \in D,\ \text{for } i = 1, \ldots, n \text{ and } Q \in \{R_1, \ldots, R_k\} \cup \{T_1, \ldots, T_l\}\ \big\}$.

With $P, \mathcal{D}$ we associate the program $P(\mathcal{D}) = P \cup \{A \longleftarrow \mid \text{ for } A \in F(\mathcal{D})\}$. For any subset I of $B(\mathcal{D})$, let $T_P(I)$ be the set of immediate consequences of I using $P(\mathcal{D})$, i.e. the set of atoms which can be deduced from I using only one application of one of the rules in $P(\mathcal{D})$. Define $T_P^0(\mathcal{D}) = F(\mathcal{D})$, $T_P^i(\mathcal{D}) = T_P(T_P^{i-1}(\mathcal{D}))$, and $T_P^\infty(\mathcal{D}) = \bigcup_{i \geq 0} T_P^i(\mathcal{D})$. Let $Q_P^i(\mathcal{D})$ be the consequences about Q in $T_P^i(\mathcal{D})$, i.e. the atoms of the form $Q(d_1, \ldots, d_n)$ in $T_P^i(\mathcal{D})$. The depth of recursion is said to be *bounded by* $b(n)$ if and only if $T_P^\infty(\mathcal{D}) = \bigcup_{0 \leq i \leq b(n)} T_P^i(\mathcal{D})$,

for all $\mathcal{D} \in \mathbb{D}_n$. The depth of recursion is said to be *bounded* if and only if it is bounded by a constant c for all n. A pair (P, Q) consisting of a *Datalog* program P together with one of its IDB predicates Q called **principal predicate** defines a query $Q_P : \mathcal{D} \mapsto Q_P(\mathcal{D})$. If no ambiguity can arise the query Q_P will simply be denoted by Q.
We will adopt a vector notation, and write $Q(\vec{x})$ instead of $Q(x_1, \ldots, x_k)$ whenever this clarifies the notations. A *proof tree* for the IDB atom $Q(\vec{x})$ serves as a witness of a computation proving $Q(\vec{x})$ from the rules of P.

Definition II.2 *A proof tree for the IDB atom $Q(x_1, \ldots, x_n)$ is a tree whose root is labeled by $Q(x_1, \ldots, x_n)$, whose internal nodes are labeled by IDB atoms, whose leaves are labeled by EDB atoms, and which is such that, if node v is labeled by $S(\vec{x})$, then the children of node v are labeled by $Q_{i_1}(\vec{x}, \vec{y}_1)$, ... , $Q_{i_r}(\vec{x}, \vec{y}_r)$ where $S(\vec{x}) \longleftarrow Q_{i_1}(\vec{x}, \vec{y}_1), \ldots, Q_{i_r}(\vec{x}, \vec{y}_r)$ is one of the rules of P defining $S(\vec{x})$. The fringe of a proof tree consists of all its leaves.*

Example II.3 For instance, letting P define the binary query TC_1 by the rules

$$P_1 \quad \begin{cases} TC_1(X,Y) \longleftarrow TC_0(X,Z)\, TC_0(Z,Y) \\ TC_0(X,Y) \longleftarrow E(X,Y) \end{cases}$$

A proof tree for $TC_1(X,Y)$ whose fringe is $E(X,Z)\ E(Z,Y)$ is depicted in Figure 1.

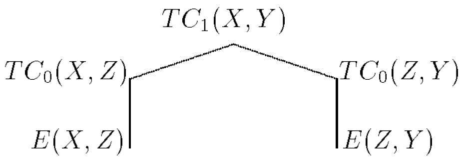

Figure 1

II.2 *DAC* Programs

Definition II.4 *The class $DAC\big(w(n), h(n)\big)$ is the set of queries Q, each query Q being expressible as follows:*
1. there exists a family $\mathcal{P}$ of Datalog programs $\mathcal{P} = \{P_n, n \in \mathbb{N}\}$, each P_n having a principal predicate Q_n of fixed arity, and such that
(i) *Q_n defines Q on databases $\mathcal{D}_n$ of size n, i.e. $Q_n(\mathcal{D}_n) = Q(\mathcal{D}_n)$,*
(ii) *P_n has width bounded by a function in $O\big(w(n)\big)$, i.e. all rule bodies of the programs P_n have at most $O\big(w(n)\big)$ atoms,*
(iii) *P_n has height bounded by a function in $O\big(h(n)\big)$, i.e. there are at most $O\big(h(n)\big)$ rules in each P_n.*
2. There exists a PTIME-computable function f which, for any n given in unary notation, produces the program P_n.

Note that if we drop the second condition (uniformity condition) in the above definition, then all monotone queries are defined with polynomial width, exponential height.

If we assume that each IDB is defined by a finite number of rules bounded by a constant k, then we may replace condition (iii) by the following weaker condition: there are at most $O(h(n))$ IDB predicates in each program P_n.

Let $DAC\big(w(n),h(n)\big)\big(b(n)\big)$ be the set of $DAC\big(w(n),h(n)\big)$ programs having depth of recursion bounded by a function in $O\big(b(n)\big)$.

$\mathcal{P} = \{P_n, n \in \mathbb{N}\}$ in $DAC\big(w(n),h(n)\big)$ ($DAC\big(w(n),h(n)\big)\big(b(n)\big)$) will be called a DAC-program (with depth of recursion $O\big(b(n)\big)$); in other words, for $\mathcal{P}$ in $DAC\big(w(n),h(n)\big)\big(b(n)\big)$, the maximum depth of a proof tree of P_n is $b(n)$, i.e. $T^{\infty}_{P_n}(\mathcal{D}) = \bigcup_{0\le i\le b(n)} T^{i}_{P_n}(\mathcal{D})$ for all $\mathcal{D} \in \mathbb{D}_n$. $b(n)$ is also called the stage function [2]. Note that the bound $b(n)$ is related to the depth parameter in circuits.

Proposition II.5 *Any query defined by a DAC-program is in $L_{\infty\omega}(\exists, \neq, \forall)$.*

Proof. We refer the reader to [4] or [6] for the definition of $L_{\infty\omega}$. Every *Datalog* program is in $L^{\omega}_{\infty\omega}(\exists)$. Let $\mathcal{P} = \{P_n, n \in \mathbb{N}\}$ be a DAC-program and let $\phi_n = Q(\vec{X})$ be a query on the principal IDB predicate of P_n; the query corresponding to $\mathcal{P}$ is defined by $\phi = \bigvee_{n\in\mathbb{N}}(card(D) = n \wedge \phi_n)$ which is a formula of $L_{\infty\omega}(\exists, \neq, \forall)$. □

II.3 Monotonicity

Recall that [3] a query is said to be **strongly monotone** if it is preserved by adding elements to the database domain, adding tuples to the database EDB relations (i.e. it is **monotone**), and moreover, it is also preserved by possibly identifying elements of the database domain. *Datalog* programs define strongly monotone queries [3]. We shall restrict our attention, here, in DAC queries which are strongly monotone.

We recall first some useful notation: Database $\mathcal{D}' = (D', R'_1, \ldots, R'_k)$ is an **extension** of database $\mathcal{D} = (D, R_1, \ldots, R_k)$, (denoted by $\mathcal{D} \subseteq \mathcal{D}'$), if $D \subseteq D'$ and for $i = 1, \ldots, k$, $R_i \subseteq R'_i$. If query Q is **monotone**, then: $\mathcal{D} \subseteq \mathcal{D}' \implies Q(\mathcal{D}) \subseteq Q(\mathcal{D}')$.

Definition II.6 *1. A DAC-program $\mathcal{P} = \{P_n, n \in \mathbb{N}\}$ defining the principal predicate Q is said to be* **monotone** *if for all $n \le m$, for all $\mathcal{D}$: $Q_{P_n}(\mathcal{D}) \subseteq Q_{P_m}(\mathcal{D})$. The monotone DAC-programs will be denoted by mDAC.*

2. A DAC-program $\mathcal{P} = \{P_n, n \in \mathbb{N}\}$ defining the principal predicate Q is said to be **strongly monotone***, denoted by $\mathcal{P} \in smDAC$, if and only if it is monotone, and moreover $\forall n, m \in \mathbb{N}$ with $n \le m$: $Q_{P_m}(\mathcal{D}_n) \subseteq Q_{P_n}(\mathcal{D}_n)$. The strongly monotone DAC-programs will be denoted by smDAC.*

Intuitively, *monotonicity* means that the P_ns are preserved under extensions of the domain and the database, in other words if a 'small' program P_n can prove the query A, then a 'large' program P_m can also prove A. *Strong monotonicity* means that the P_ns are preserved under extensions of the domain and the database, and moreover, if for $n \le m$, a query A can be proven on domain D_n using the program P_m, then it can also be proven using the program P_n. In other words, a strongly monotone program is monotone, and moreover, if a 'large' program P_m can prove the query A on a 'small' database, then a 'small' program P_n can also prove A. Note that the (strong) monotonicity of a DAC-program with principal predicate Q is *a priori* a notion different from the (strong) monotonicity of query Q; our terminology is however consistent because (see Proposition II.10) a DAC-program with principal predicate Q is strongly monotone if and only if the corresponding query Q is strongly monotone.

PROPOSITION II.7 *If the DAC-program $\mathcal{P} = \{P_n, n \in \mathbb{N}\}$ defining the principal predicate Q is monotone, then for all $n \leq m$, for all $\mathcal{D}, \mathcal{D}'$:* $\quad \mathcal{D} \subseteq \mathcal{D}' \implies Q_{P_n}(\mathcal{D}) \subseteq Q_{P_m}(\mathcal{D}')$.

Let $DAC^{\neq}$ (resp. $Datalog^{\neq}$) be the language obtained by adjoining a symbol $\neq$ to our basic language DAC (resp. $Datalog$). The following examples offer some more intuition on DAC queries:

EXAMPLE II.8 1. Consider a graph as a database with a single binary relation, and the query $Q(x,y) = \{(x,y) \mid \text{there exists a path of length } \lfloor \log n \rfloor \text{ between } x \text{ and } y\}$. This query is $DAC(\log n, 1)(1)$ but it is not $Datalog$ definable because it is not strongly monotone: indeed, let $n = 2^k$ and let $\mathcal{D}_n$ be a database of size n, then there may be a path of length $\lfloor \log n \rfloor = k$ between x and y, but if we collapse 2 points of $\mathcal{D}_n$, we obtain a database $\mathcal{D}_{n-1}$ of size $n-1$ and there may no longer be a path of length $\lfloor \log(n-1) \rfloor \leq k-1$ between x and y. Because query Q coincides with the query Q_1^0 of Theorem IV.5, query Q is also $DAC(1, \log\log n)(\log\log n)$.

2. On the previous class of graphs with a single binary relation,

- parity, i.e. the query which is true iff $\mathcal{D}$ has an even number of elements, is a non monotone $DAC(1,1)(1)$ query: take the trivial program that always accepts or always rejects according to the parity of n,
- connectivity, i.e. the query which is true iff for any d, d' in $\mathcal{D}$ there is a path from d to d', is not a DAC query: the argument applied to show that it is not $Datalog$ (see [1]) can also be applied to show that it is not DAC.

3. The query Q^k defined by

$$Q^k(\mathcal{D}) = \begin{cases} true & \text{if } \mathrm{card}(\mathcal{D}) \geq k \\ false & \text{if } \mathrm{card}(\mathcal{D}) < k \end{cases}$$

is a monotone $DAC(1,1)$ query, defined by the DAC-program $\mathcal{P}^k = \{P_n^k, n \in \mathbb{N}\}$ with

$$P_n^k = \begin{cases} Q^k \longleftarrow & \text{for } n \geq k \\ \square & \text{for } n < k \end{cases}$$

$\mathcal{P}^k$ is monotone $DAC(1,1)$ but not strongly monotone, since for $n < k \leq m$, $Q_{P_m^k}(\mathcal{D}_n) = true$, but $Q_{P_n^k}(\mathcal{D}_n) = false$. Note that query Q^k is expressible in $Datalog^{\neq}$, but it is not expressible in $Datalog$, because it is not strongly monotone.

II.4 General properties of DAC-programs

PROPOSITION II.9 $smDAC(1,1) = Datalog$.

It can be shown that strongly monotone DAC-programs correspond to strongly monotone queries.

PROPOSITION II.10 *A strongly monotone DAC-program $\mathcal{P} = \{P_n, n \in \mathbb{N}\}$ defines a strongly monotone query. Conversely, if a DAC-program $\mathcal{P}$ defines a strongly monotone query, then $\mathcal{P}$ is strongly monotone.*

The following proposition shows a trade-off between the parameters of a DAC-program.

PROPOSITION II.11

$$DAC(w(n), h(n))(b(n)) \subseteq DAC\big(1, h(n)(w(n)-1)\big)\big(b(n)\log w(n)\big)$$

The idea is to use a 'divide-and-conquer' strategy to simulate one rule of width w by $\log w$ rules of width 2. In other words, DAC allows for trade-offs between width, height and recursion depth:

- Proposition II.11 shows that a rule of width $w(n)$ can be simulated by $w(n)-1$ rules defining new IDBs of width 2 at the cost of an explosion in the number of arguments of the predicates;
- similarly, recursion depth could be decreased at the cost of increasing the number of IDB predicates together with the number of arguments of the predicates.

COROLLARY II.12 *1.* $DAC(w(n),1)(1) \subseteq DAC(1,w(n))(\log w(n))$ *and*
2. $DAC(1,n)(n) \not\subseteq DAC(n,1)(1)$.

The above corollary asserts that width can be simulated by height (and number of IDB predicates), but that the converse is false.

II.5 Bounded and strongly monotone DAC

Let n be the number of elements (entities) in the database, and N be the number of ground facts in the database; assuming every database has a fixed number p of EDB relations of maximum arity k, then $N \leq pn^k$ and that implies $O(\log n) = O(\log N)$. Hence, whenever we are interested in computations of at most logarithmic complexity, the complexity can be evaluated with respect to n or N indifferently.

A DAC-program is said to be **bounded** if its depth of recursion is bounded by a finite constant b, i.e. all computations of the programs P_n stop after at most b recursive calls, for all databases, and all n.

PROPOSITION II.13 *The classes Datalog and* $mDAC(1,1)(1)$ *are incomparable.*

Proof. 1. Note that (bounded) $mDAC(1,1)(1) \not\subseteq$ *Datalog*: this is shown by the query Q^k: $\mathrm{card}(\mathcal{D}) \geq k$ (of Example II.8 3)), which is monotone and bounded DAC, but is not *Datalog*, because it is not strongly monotone.

2. Conversely, *Datalog* $\not\subseteq$ $mDAC(1,1)(1)$, this follows from Example I.1 (iv). □

THEOREM II.1 *For any function* h, $smDAC(1,h(n))(1)$ *coincides with bounded Datalog.*

III The expressiveness of DAC-programs on chain queries

Queries defining paths on databases consisting of graphs are quite useful in separating various classes of DAC-programs.

Let $\mathcal{D}_n = (D_n, R)$ be a database of type 2 on a domain D_n consisting of n vertices, i.e. our databases $\mathcal{D}_n$ will have a single edge relation R. On such a database, we will define some queries. We first give some useful definitions.

• *Paths* – On a database consisting of a graph given by an edge relation R, let $p = u_1 \cdots u_n$ be a path from x to y, i.e. a sequence such that for $k = 1, \ldots, n$, $u_k = (x_k, x_{k+1}) \in R$ and $x_1 = x$, $x_{n+1} = y$.

• *Paths accepted by queries or programs* – Let p be a path from x to y. Let Q be a binary query: p is said to be *accepted by the query* Q if and only if $(x, y) \in Q(\mathcal{D}_n)$.

Definition III.14 *Let $f(x)$ be a strictly increasing function on $\mathbb{N}$. We define the chain query Q_f by:*

$$Q_f = \{(x,y) \mid \text{there is a path of length } f(q) \text{ from } x \text{ to } y \text{ for some } q \in \mathbb{N}\}. \quad (1)$$

Such chain queries are completely described by languages $L \subset \{R\}^*$, or subsets of $\mathbb{N}$: to the query Q_f corresponds the language $L_f = \{R^{f(q)} \mid q \in \mathbb{N}\}$, or the subset $\{f(q) \mid q \in \mathbb{N}\}$ of $\mathbb{N}$. Note that chain queries are strongly monotone: they are clearly monotone, and if some elements of the database are collapsed, the edges connecting them are not deleted, but possibly give rise to cycles, hence the lengths of the paths are preserved.

III.1 Pumping Lemma

Let $\mathcal{D} = (D, R_1, \ldots, R_k)$ be a database of type $a = (2, \ldots, 2)$ on a domain D consisting of n vertices, i.e. all the R_is are binary relations. Let P be a *Datalog* program with principal predicate Q. Consider the fringe e of a proof tree for $Q(X,Y)$ (see Definition II.2). A fringe e which is a path of the form $R_{i_1}(x, x_1) \ldots R_{i_p}(x_{p-1}, y)$ is called a *word fringe* and $\omega = R_{i_1} \ldots R_{i_p}$ is said to be the corresponding word over the EDB symbols; we say that P *accepts* word ω via fringe e; if moreover ω can be pumped in the sense of the context–free language pumping lemma, keeping the size l of the pumped part less than N, i.e. $0 < l < N$, ω is said to *have a pumping string of length less than* N. In [3], the following technical result is proven:

Lemma III.15 *Let P be a Datalog program, and assume that P accepts word ω via a fringe. There exists an integer N, depending only on P, such that, if ω is longer than N, then ω has a pumping string of length $< N$.*

Let P be a Datalog program with the following parameters: ν is the number of IDB predicate symbols, w is the maximum width (=number of arguments) of IDB predicate symbols, d is the maximum number of occurrences of IDB predicate symbols per rule, μ is the maximum number of EDB predicate symbols per rule. Let $N = \mu d^{\nu w^{4(w!3^w)^2}}$ and let ω be a word accepted by P via a fringe. If ω is longer than N, then ω has a pumping string of length $< N$.

Moreover, if we assume that all other parameters except ν are bounded from above by c_0 and $c' = c_0^{4(c_0!3^{c_0})^2}$, then we can take $N = c_0^{1+\nu c'}$. Consider, now, a strongly monotone $DAC(1, h(n))(b(n))$-program $\mathcal{P}$. For each P_n of $\mathcal{P}$ the above lemma holds if we can guarantee that ω_n (i.e., now the word has to be of length n) is longer than $N = c_0^{1+h(n)c'} = c2^{ch(n)}$, where c_0, c', c are constants of $\mathcal{P}$ (i.e., not depending on n). This is the case if $h(n) \in O(\log n)$. In addition, we need strong monotonicity to guarantee that P_n gives the right answer also if applied to databases smaller than n. Hence the following pumping lemma:

Lemma III.16 *Let $h(n)$ be a function on $\mathbb{N}$; assume that $h(n) \in O(\log n)$. Let $\mathcal{P}$ be a strongly monotone $smDAC(1, h(n))$-program, and let $l(i)$ be an increasing function on $\mathbb{N}$. If $\mathcal{P}$ accepts the query Q_l, then there exist natural numbers i_0 and c such that, for any natural number $i > i_0$, $l(i+1) - l(i) \le c2^{ch(l(i+1))}$, where c is a function of the constant parameters of $\mathcal{P}$.*

III.2 Applications

We use Lemma III.16 to separate DAC families by showing that 'too small' programs cannot capture chain queries computing paths whose length grow 'too fast'.

Example III.17 The perfect square query Q_f defined by (1) (see Definition III.14) where $f(q) = q^2$, is strongly monotone and PTIME; it is not *Datalog* ([3]), but we will see that it is $smDAC(\log n, n)(\log n)$ (cf. Corollary III.18).

Theorem III.2 *Let Q_f be defined as in Definition III.14 with f PTIME-computable. Assume that, for any database $\mathcal{D}_n$ on a domain D_n of size n, if there exists a path of length $f(q)$ between two elements x and y of D_n, then there exists a path of length $f(q') \leq 2^{g(n)}$. Then Q_f is in the class $DAC(g(n), 2^{g(n)})(g(n))$.*

Proof. A program using a 'divide-and-conquer' strategy can accept the paths of length 2^i for $i \leq g(n)$ in recursion depth $g(n)$ (see Example I.1 (i)). Then we can combine paths of length 2^i for each bit i of the binary representation of $f(n)$. □

So this means for instance that each Q_f corresponding to an $f(n)$ with $|f(n)| < n$ is expressible in $smDAC(\log n, n)(\log n)$.

We first study the perfect p-th power query Q_p defined by (1) (see Definition III.14) where $f(q) = q^p$.

Corollary III.18 *The perfect p-th power query is in $smDAC(\log n, n)(\log n)$.*

Theorem III.3 *The perfect p-th power query Q_p (i) is expressible in $smDAC(1, n^{12})(\log n)$, (ii) is not expressible in $smDAC(1, (\log n)^{1-\varepsilon})(\log n)$ for $\varepsilon > 0$, and (iii) is not expressible in Datalog.*

Proof. (i) It has been shown in [3] that, if a directed graph with n vertices has a 'long' path from x to y whose length is a perfect p-th power, then it also has a 'short' path of length l' a perfect p-th power with $l' \leq n^6$; hence it suffices to find all paths of length l'^p, with $l' \leq n^6$ to answer the query. We then apply (with some care) the above Theorem III.2 and Proposition II.11. Non-expressibility follows from Lemma III.16 for (ii) and from [3] for (iii). □

We now study the Q_{2^*} query, namely the query Q_f as defined in Definition III.14, where $f(x) = 2^x$.

Proposition III.19 *Assume $\mathcal{D}_n$ is a size n database containing m loops of lengths $i_1, \ldots, i_k$ with $k \leq n$, k fixed. The perfect power-of-two query defined by $Q_{2^*}(x, y)$ if and only if there exist an integer q and a path of length 2^q from x to y is in $DAC(1, poly)(\log n)$, and is not in $DAC(1, (\log n)^{1-\varepsilon})(\log n)$.*

IV A hierarchy of DAC-programs

Let $\mathcal{D}_n = (D_n, R)$ be a database consisting of a domain D_n of n vertices with a single edge relation R. On $\mathcal{D}_n$, consider the query $Q_0 = \{(x, y) |$ there is a path of length n from x to $y\}$. Using this query, we show that the following exponential hierarchy is strict.

THEOREM IV.4 *We have, for any $i > 0$, the strict inclusion:*

$$DAC(1, \underbrace{\log\log\ldots\log}_{i \text{ times}} n)(\underbrace{\log\log\ldots\log}_{i \text{ times}} n) \subset DAC(1, \underbrace{\log\log\ldots\log}_{i-1 \text{ times}} n)(\underbrace{\log\log\ldots\log}_{i-1 \text{ times}} n)$$

In between any two levels of the exponential hierarchy of Theorem IV.4, we can squeeze a strict polynomial subhierarchy, namely we have strict sublogarithmic polylog hierarchies as stated in the following

THEOREM IV.5 *Let $n_0 = n$, and, for $i > 0$, $n_i = \underbrace{\log\log\ldots\log}_{i \text{ times}} n$; we have, for any $i \geq 2$ and $k \geq 1$:*

$$\begin{aligned} DAC(1, n_i)(n_i) \subset \cdots \subset DAC(1, n_i^k)(n_i^k) \\ \subset DAC(1, n_i^{k+1})(n_i^{k+1}) \subset \cdots \subset DAC(1, n_{i-1})(n_{i-1}) . \end{aligned}$$

V Conclusion

We introduced the class DAC of queries definable by a class of *Datalog* programs with various restrictions on the width, height and depth. We first showed that bounded $smDAC(1, h(n))$, for any h, coincides with bounded *Datalog* and we studied the class $DAC(1, poly)(\log n)$: it contains many natural queries which are not *Datalog* expressible.

We then proved non-expressiveness results for the class $DAC(1, \log\log n)(\log n)$ with the pumping technique and we inferred various strict hierarchies. DAC-definability of queries may turn out to be interesting in practice: we can compute transitive closure by a DAC-program of recursion depth $\log n / \log\log n$ as opposed to the usual $\log n$ (see Example I.1 (iii)) and other gains may be possible for other queries.

Acknowledgments: We thank Patrick Cegielski, Yuri Gurevich, Miklos Santha and Dan Suciu for interesting discussions related to this paper.

VI References

[1] S. Abiteboul, R. Hull, V. Vianu, *Foundations of databases*, Addison-Wesley (1995).

[2] S. Abiteboul, P. Kanellakis, Deciding bounded recursion in database logic programs, *SIGACT News* 20 (1989), 17–23.

[3] F. Afrati, S. Cosmadakis, M. Yannakakis, On *Datalog* vs. polynomial time, *J. Comput. Syst. Sc.* **51** (1995), 177–196.

[4] J. Barwise, The Syntax and Semantics of Infinitary Languages, Springer-Verlag (1968).

[5] R. Boppana, M. Sipser, The complexity of finite functions, *Handbook of Theoretical Computer Science* **14** (1990), 757–804.

[6] H. Ebbinghaus, J. Flum, *Finite Model Theory*, Springer-Verlag (1995).