A Query Language for NC
(Extended Abstract)

Dan Suciu and Val Breazu-Tannen*

University of Pennsylvania
email: [suciu,val]@cis.upenn.edu

Abstract. We show that a form of divide and conquer recursion on sets together with the relational algebra expresses exactly the queries over ordered relational databases which are NC-computable. At a finer level, we relate k nested uses of recursion exactly to AC^k, $k \geq 1$. We also give corresponding results for complex objects.[2]

1 Introduction

NC is the complexity class of functions that are computable in polylogarithmic time with polynomially many processors on a *parallel random access machine* (PRAM). The query language for NC discussed here is centered around a *divide and conquer recursion* (*dcr*) on sets which has obvious potential for parallel evaluation and can easily express, for example, transitive closure and parity. *dcr* with parameters e, f, u defines the unique function $\varphi = dcr(e, f, u)$ such that:

$$\begin{aligned} \varphi(\emptyset) &\stackrel{\text{def}}{=} e \\ \varphi(\{y\}) &\stackrel{\text{def}}{=} f(y) \\ \varphi(s_1 \cup s_2) &\stackrel{\text{def}}{=} u(\varphi(s_1), \varphi(s_2)) \text{ when } s_1 \cap s_2 = \emptyset \end{aligned}$$

For parity, we take $e \stackrel{\text{def}}{=} \mathit{false}$, $f(y) \stackrel{\text{def}}{=} \mathit{true}$ and $u(v_1, v_2) \stackrel{\text{def}}{=} v_1 \ \mathit{xor} \ v_2$. To compute the transitive closure of some binary relation r, take $e \stackrel{\text{def}}{=} \emptyset$, $f(y) \stackrel{\text{def}}{=} r$ and $u(r_1, r_2) \stackrel{\text{def}}{=} r_1 \cup r_2 \cup r_1 \circ r_2$. Then, the transitive closure of r is obtained by applying φ to the set of nodes of the relation r, namely $tc(r) = \varphi(\Pi_1(r) \cup \Pi_2(r))$, where Π_1, Π_2 are the relational projections. In general, $dcr(e, f, u)$ is well-defined when there is some set containing e and the range of f, on which u is associative, commutative and has the identity e. For parity, this is the set $\mathbb{B}$ of booleans, while for transitive closure, it is the set $\{r \cup r^2 \cup \ldots \cup r^n \mid n \geq 0\}$.

* The authors were partially supported by NSF Grant CCR-90-57570 and ONR Contract NOOO14-93-11284

[2] These results were first announced at PODS'94 [30] whose proceedings contain a somewhat more detailed exposition than the one below. A journal version is in preparation. Here we are giving just enough definitions to clarify the statement of the main results. See also `http://www.cis.upenn.edu/~wfan/dbl.html`

We show that *dcr* together with the relational algebra expresses exactly the queries over *ordered* databases of flat relations that are NC-computable. We also show that a bounded version of *dcr* together with the nested relational algebra expresses exactly the queries over *ordered* databases of complex objects that are NC-computable. In fact, we prove the more refined versions that relate k nested uses of (bounded) *dcr* exactly to the subclass AC^k of NC where $k \geq 1$. Some explanations are in order:

- Computable queries are in the sense of Chandra and Harel [10], naturally extended to complex objects.

- Any language that can express the class of queries expressed by first-order logic would do just as well as the relational algebra. Similarly for complex objects, where a corresponding class of tractable queries has emerged from several equivalent formalisms. Some of these formalisms are syntactically restricted higher-order logics, others are algebraic languages, often called nested relational algebras, hence our phrasing above.

- *dcr* and (nested) relational algebra have meaning over any (nested) relational database. But, as with all known characterizations of query complexity classes below NP, we know how to capture the entire NC only over ordered databases. Formally, we do this by extending the language with an order predicate.

- A bounded version of *dcr* is necessary over complex objects, otherwise queries of high complexity such as *powerset* will be expressible. The bounded version is obtained by intersecting the result with a bounding set at each recursion step (section 2). This is similar to the bounded fixpoints studied in [29], and, as with fixpoints, over flat relations *dcr* can always be expressed through bounded *dcr* (section 2).

We believe that these results are of interest from two angles.

∠ **Query language design.** *dcr* is a well-known construct. It appears under the name *pump*, in a language specifically designed for a parallel database machine, FAD [4]. Following FAD, but under the name *hom*, it was included in Machiavelli [24] where it fit nicely into the language's type system. Called (a form of) *transducer*, it is part of SVP [27], precisely in order to support divide and conquer parallelism. Some limitations of its theoretical expressive power were examined (under the name *hom*) by Immerman, Patnaik, and Stemple ([22] theorem 7.8). They also note that *dcr* is in NC.

As part of a larger group of researchers, we became interested in *dcr* because it fits into a natural hierarchy of query languages that share a common semantic basis, built around forms of structural recursion on collection types [7, 6, 8] (see section 2). Theoretical studies of expressiveness, such as [33, 6, 29] and the present paper help us with the choice and mix of primitives, as well as implementation strategies. In particular, *dcr* is at the core of a sublanguage for which we are currently seeking efficient implementation techniques for a variety of parallel architectures.

∠ **Computational complexity.** Following Vardi and Immerman's influential result [32, 18] that first-order logic with least fixed point captures exactly the $PTIME$-computable queries on flat relations over ordered databases, several characterizations of low complexity classes in terms of logics or algebras used in databases have been discovered with the hope that logical methods may give insights into the difficult problem of complexity class separation. We mention first a few of these characterizations which have had a direct influence on the work here.

For parallel complexity classes, Immerman [21] shows that the class of finite and ordered relational structures recognizable in parallel time $t(n)$ (n is the size of the structure) on a certain CRCW (concurrent read - concurrent write) PRAM coincides with the class of structures definable by a first-order induction [23] of depth up to $t(n)$. For complex object databases, Grumbach and Vianu [16, 15] give a syntactic restriction of the ramified higher-order logic CALC which, together with inflationary fixpoints and in the presence of order, captures exactly the $PTIME$-computable complex-object queries. Suciu [29] shows that, in the presence of order, the same class of queries is captured by the nested relational algebra augmented with an inflationary bounded fixpoint operator.

To the best of our knowledge, no characterization of parallel complexity classes of queries over complex objects has been given before. What is more likely to set our results apart, however, is the *intrinsic* nature of the language we are proposing: the semantics of *dcr* puts it naturally in NC; there is no need to impose logarithmic bounds on the number of iterations or recursion depth. Moreover, it can be shown that a different kind of recursion on sets, namely structural recursion on the insert presentation of sets ([7]; notation *sri*; definitions reviewed in section 2), together with the relational algebra expresses exactly the $PTIME$-computable queries on ordered databases. This follows from results in [22]; we state the corresponding result for complex objects in proposition 8. Hence, at least over ordered databases, the difference between NC and $PTIME$ boils down to two different ways of recurring on sets, divide and conquer vs. element by element.

Gurevich [17] and Compton and Laflamme [12] characterize the $DLOGSPACE$- and respectively the NC^1-computable global functions on finite and ordered relational structures as algebras with certain primitive recursion schemas. Compton and Laflamme capture NC^1 also with first-order logic augmented with BIT. and with an operator for *defining relations by primitive recursion*. The kinds of recursions used in these two papers are very different from *dcr* because they depend on some linear ordering of the underlying structures for their actual definition. While *dcr* is a form of recursion on finite sets, these recursions are on notations for elements of (linearly ordered) finite sets. Of course, we do not attempt to characterize $DLOGSPACE$ or NC^1 or, for that matter, any class below AC^1, but see Immerman's characterizations of such classes in terms of languages more in the spirit of ours than of those of Gurevich, Compton, and Laflamme [20, 19].

We should also mention here the work of Clote [11] for related characterizations of most parallel complexity classes of *arithmetical* functions.

We should point out, however, one sense in which our language is not as neat as, say, first-order logic with least fixpoint, which captures $PTIME$ in the presence of order, or first-order logic with transitive closure, which captures *NLOGSPACE* in the presence of order. For *dcr* to be well-defined, the operations involved in it must satisfy certain algebraic identities (associativity, commutativity, identity) and this turns out to be an undecidable condition (in fact Π^0_1 complete; see section 2). Of course, only a certain family of instances of *dcr* is actually needed in the simulations, and for these, the algebraic conditions always hold. Hence, it is of theoretical interest that there is a decidable sublanguage of *dcr* plus relational algebra which captures exactly *NC* in the presence of order. In practice, we have found it useful to provide special syntax for some instances of *dcr* in which the algebraic conditions are automatically satisfied, but we found it counterproductive to limit *dcr* to these instances, as other uses kept appearing.

2 Recursion on Sets

A fruitful approach to the design of query languages for complex objects is to consider tuples and sets as orthogonal [6, 8]. Hence, there will be primitives that work on tuples, primitives that work on sets, and general primitives for combining other primitives. In this section we discuss several choices for defining recursions on sets and the relationship between them. One of these forms of yrecursion—*dcr*—will be at the core of our "query languages for *NC*". A precise definition of the languages will be given in section 3.

Complex objects are built essentially from tuples and finite sets. To describe them, we define the **complex object types** by the grammar:

$$t \stackrel{\text{def}}{=} \mathbb{D} \mid \mathbb{B} \mid \mathit{unit} \mid t \times t \mid \{t\}$$

where $\mathbb{D}$ is some base type, $\mathbb{B}$ is the type of booleans and *unit* is the type containing only the empty tuple ($\mathit{unit} = \{()\}$). The values of type $s \times t$ are pairs (x, y) with $x \in s, y \in t$, and the values of type $\{t\}$ are finite sets of elements from t. Products of types of the form $\{s\}$, with s a product of base types ($\mathbb{D}, \mathbb{B}, \mathit{unit}$), are called **flat types**. We will also need to consider **function types** having the form $s \to t$, where s and t are complex object types.

There seem to be two basic ways of describing the structure of finite sets. In one way, they are generated by finitely many (maybe zero!) binary unions of singleton sets. We call this the *union presentation*. In another way, they are generated by finitely many insertions of one element, starting with the empty set. We call this the *insert presentation*. Recognizing the relevant algebraic identities satisfied by union (associativity, commutativity, idempotence, has $\emptyset$ as an identity) and by element insertion (i-commutativity and i-idempotence) gives us two different algebraic structures on finite sets. Both these algebras are characterized by universality properties, which amount to definitions of functions by *structural recursion* [7, 6]. We have a structural recursion construct on the union presentation, *sru*:

$$\frac{e : t \quad f : s \to t \quad u : t \times t \to t}{\mathit{sru}(e, f, u) : \{s\} \to t}$$

$$
\begin{aligned}
sru(e,f,u)(\emptyset) &\stackrel{\text{def}}{=} e \\
sru(e,f,u)(\{y\}) &\stackrel{\text{def}}{=} f(y) \\
sru(e,f,u)(s_1 \cup s_2) &\stackrel{\text{def}}{=} u(sru(e,f,u)(s_1), \\
&\qquad\quad sru(e,f,u)(s_2))
\end{aligned}
$$

$sru(e,f,u)$ is well-defined when there is some subset of t containing e and the range of f, on which u is associative, commutative, idempotent, and has the identity e. We also have a structural recursion construct on the insert presentation, $sri(x \frown s)$ is the element insertion operation, $\{x\} \cup s$):

$$
\frac{e : t \quad i : s \times t \to t}{sri(e,i) : \{s\} \to t}
$$

$$
\begin{aligned}
sri(e,i)(\emptyset) &\stackrel{\text{def}}{=} e \\
sri(e,i)(y \frown s) &\stackrel{\text{def}}{=} i(y, sri(e,i)(s))
\end{aligned}
$$

$sri(e,i)$ is well-defined when i is *i-commutative*, and *i-idempotent* ($i(x,i(y,s)) = i(y,i(x,s))$, $i(x,i(x,s)) = i(x,s)$) on some subset of t containing e.

The *divide and conquer recursion dcr*, whose definition was already given in section 1, is superficially related to sru. For example $sru(e,f,u)$ is well-defined then so is $dcr(e,f,u)$ and they are equal. But dcr is potentially more expressive, since u need not be idempotent. In fact, we do not know if sru can express parity or transitive closure. One can prove:

Proposition 1. *sri can express dcr, which in turn can express sru [7, 30]. All these are done with at most polynomial overhead.*

Over complex objects dcr can express *powerset* hence we need some restriction if we are to stay within NC. An analog to Peter Buneman's idea of *bounded* fixpoints [29] does the job. We define a **PS-type** (product of sets) to be either a set type, or a product of PS-types. The **bounded** version of dcr is defined by:

$$
\frac{e : t \quad f : s \to t \quad u : t \times t \to t \quad b : t}{bdcr(e,f,u,b) : \{s\} \to t}
$$

where t is a PS-type, with the semantics:

$$
bdcr(e,f,u,b) \stackrel{\text{def}}{=} dcr(e \cap b, f \cap b, u \cap b)
$$

(here $(u \cap b)(y,y') \stackrel{\text{def}}{=} u(y,y') \cap b$, etc). As for dcr, we define the **bounded** sri, $bsri(e,i,b)$, for some PS-type t, to be $sri(e \cap b, i \cap b)$. Proposition 1 easily extends to the bounded versions of recursion. Over flat relations the explicit bounding is unnecessary:

Proposition 2. *bdcr together with the relational algebra can express dcr when its arguments are flat relations and its values are of flat PS-type. Similarly for bsri and sri.*

Unfortunately, for a language at least as expressive as first-order logic, verifying most of the identities required by the *sri*, *dcr*, and *sru* constructs is as hard as testing the validity of a first-order formula in all finite models, hence it is a Π^0_1-complete question.

3 Query Languages

In this section we define the languages that offer characterizations of NC over complex objects and over flat relations. For complex objects, the language is obtained by adding *bdcr* and an order predicate to the *nested relational algebra* $\mathcal{NRA}$. This name can be used to describe a language that has the same expressive power as Schek and Scholl's NF^2 relational algebra [28], or Thomas and Fischer's algebra [31], or Paredaens and Van Gucht's *nested algebra* [25, 26], or Abiteboul and Beeri's *algebra without powerset* [2]. The particular formalization we use here was essentially introduced in [8] and is based on what is called there the **monad calculus**.

For each type s we assume an infinite set of variables to be given, and write x^s for a variable of type s. As usual, we distinguish between free and bound variables, and we identify those expressions that differ only in the name of some bound variables. $\mathcal{NRA}$ is defined by the rules in figure 1.

$$\frac{}{x^t : t} \qquad \frac{e_1 : t_1 \quad e_2 : t_2}{(e_1, e_2) : (t_1, t_2)}$$

$$\frac{e : t_1 \times t_2}{\pi_1(e) : t_1} \qquad \frac{e : t_1 \times t_2}{\pi_2(e) : t_2}$$

$$\frac{}{\emptyset : \{t\}} \qquad \frac{e : t}{\{e\} : \{t\}} \qquad \frac{e_1 : \{t\} \quad e_1 : \{t\}}{e_1 \cup e_2 : \{t\}}$$

$$\frac{e_1 : \mathbb{D} \quad e_2 : \mathbb{D}}{e_1 = e_2 : \mathbb{B}} \qquad \frac{}{() : unit}$$

$$\frac{e : \{t\}}{empty(e) : \mathbb{B}} \qquad \frac{e : \mathbb{B} \quad e_1 : t \quad e_2 : t}{if\ e\ then\ e_1\ else\ e_2 : t}$$

$$\frac{e : t}{\lambda x^s.e : s \to t} \qquad \frac{f : s \to t \quad e : s}{f(e) : t}$$

$$\frac{f : s \to \{t\}}{ext(f) : \{s\} \to \{t\}}$$

Fig. 1. The Nested Relational Algebra

We briefly describe the semantics of the expressions: (e_1, e_2) constructs a tuple, π_1, π_2 are the projections, $\{e\}$ is the singleton set, $empty(e)$ returns true iff $e = \emptyset$, *if* e *then* e_1 *else* e_2 equals e_1 iff $e = true$ and e_2 otherwise, $\lambda x^s.e$ denotes a function whose input is the variable x^s, $f(e)$ is function application, and $ext(f)(\{x_1, \ldots, x_n\}) \stackrel{\text{def}}{=} f(x_1) \cup \ldots \cup f(x_n)$. A possible set Σ of **external functions** $p : dom(p) \to codom(p)$ could be added to the language; in this case, we denote the language by $\mathcal{NRA}(\Sigma)$. We abbreviate with $\lambda(x, y).e$ the expression $\lambda z.e[\pi_1(z)/x, \pi_2(z)/y]$.

$\mathcal{NRA}$ is powerful enough to express the following functions: set difference, set intersection, cartesian product, database projections, equalities at all types, selections over predicates definable in the language, nest and unnest [8].

We will make two extensions to $\mathcal{NRA}$ in order to obtain a language that characterizes NC. One extension is the addition of divide-and-conquer recursion on sets. *dcr* is too powerful on complex objects since it can express *powerset* (actually one can show that $\mathcal{NRA}(dcr)$ has the same expressive power as Abiteboul and Beeri's *algebra* [2]). So instead, we will add the bounded version of this kind of recursion and consider the language $\mathcal{NRA}(bdcr)$. We shall have occasion to contrast this languages with $\mathcal{NRA}(bsri)$. Note that proposition 1 states that $\mathcal{NRA}(bdcr) \subseteq \mathcal{NRA}(bsri)$, and this holds even in the presence of external functions.

The second extension concerns *ordering*. All known characterizations of complexity classes below $PTIME$ are about ordered databases, and ours is no exception. We isolate the use of ordering into the addition of an external function $\leq : \mathbb{D} \times \mathbb{D} \to \mathbb{B}$, which is understood always to denote a linear order on $\mathbb{D}$. The order relation can be lifted to all types.

We have thus reached the language $\mathcal{NRA}(bdcr, \leq)$ which is the subject of one of our main theorems, characterizing the NC-computable queries on ordered databases of complex objects (theorem 3).

In the same theorem, we obtain a finer characterization, involving the AC-hierarchy, for which we need the notion of **the depth of recursion nesting** $depth(e)$, of some expression e. We define this to be to be the maximum depth of recursions occurring in e. For example, for *bdcr*: $depth(bdcr(e, f, u)) \stackrel{\text{def}}{=} \max(depth(e), depth(f), 1 + depth(u))$ (only u is actually iterated). We denote $\mathcal{NRA}(bdcr^{(k)})$ the restrictions of the language $\mathcal{NRA}(bdcr)$ to iteration depth $\leq k$. We have thus obtained the hierarchy of languages $\mathcal{NRA}(bdcr^{(k)}, \leq)$ which is the subject of the finer characterization given in theorem 3.

To characterize the NC-computable queries over ordered flat relations (theorem 4), we denote by $\mathcal{NRA}^1$ the restriction of $\mathcal{NRA}$ to types of "set height ≤ 1", i.e. the only types allowed in $\mathcal{NRA}^1$ as inputs, outputs and intermediate types are products of base and flat types. Its expressive power is essentially the same as that of the relational algebra [6], hence first-order logic. We make the same two extensions as for $\mathcal{NRA}$, except that here it is safe to use *dcr*. For good measure, proposition 2 states that it doesn't matter: $\mathcal{NRA}^1(bdcr) = \mathcal{NRA}^1(dcr)$. Note however that this fails in the presence of certain external functions. Again we will have occasion to contrast $\mathcal{NRA}^1(dcr)$ with $\mathcal{NRA}^1(sri)$ so note that proposition 1 states that $\mathcal{NRA}^1(dcr) \subseteq \mathcal{NRA}^1(sri)$, and this holds even in the presence of external functions. Hence, the languages of interest for flat relations are $\mathcal{NRA}^1(dcr, \leq)$ and the hierarchy $\mathcal{NRA}^1(dcr^{(k)}, \leq)$.

4 Main Results

Assuming some enumeration of the base type $\mathbb{D}$ to be given, we encode complex objects as strings over the alphabet $A = \{0, 1, \{, \}, (,), comma, blank\}$. The order of elemenets in the sets is irrelevant, but no duplicates are allowed. E.g. the object $\{(a, (b, a)), (c, (a, a)), (a, (b, c))\}$ could be encoded as the string $\{1, (10, 1), (11, (1, 1)), (1, (10, 11))\}$, if the encoding of $\mathbb{D}$ assigns $1, 2, 3$ to a, b, c respectively. Blanks may be arbitrarily scattered in an encoding, but not insided the binary numbers. Furthermore, we make a second encoding of complex objects as binary strings (i.e. in $\{0, 1\}^*$) by replacing each of the 8 symbols in A with a 3-bits binary number.

Recall that AC^k is the class of functions $\{0, 1\}^* \to \{0, 1\}^*$ computable by a "uniform" family of circuits of polynomial size and of depth $O(\log^k n)$ [13]. We define Q-NC and CQ-NC to be the class of queries over base types and flat relations, and complex objects respectively, which are in NC. Similarly, we define the subclasses Q-AC^k and CQ-AC^k.

Theorem 3. *$\mathcal{NRA}(bdcr, \leq) = CQ$-$NC$. More precisely, for every $k \geq 1$ we have $\mathcal{NRA}(bdcr^{(k)}, \leq) = CQ$-$AC^k$.*

Theorem 4. *$\mathcal{NRA}^1(dcr, \leq) = Q$-$NC$. More precisely, for every $k \geq 1$ we have $\mathcal{NRA}^1(dcr^{(k)}, \leq) = Q$-$AC^k$.*

These languages are purely for complex objects, respectively relations. But many external functions of practical interest such as the usual arithmetical operations $(+, *, -, /, etc)$, and the usual aggregate functions (cardinality, sum, average, etc.) are also in NC. Can they be added in? The answer is given by:

Proposition 5. *Let Σ be an extension consisting of possible additional base types and a set of functions computable in NC. Then $\mathcal{NRA}(\Sigma, bdcr) \subseteq NC$. However, $\mathcal{NRA}^1(\mathbb{N}, +, dcr)$ can express exponential space queries.*

Immerman in [21] and Barrington, Immerman and Straubing in [5] prove that FO is included in FO-DCL-uniform AC^0, and that FO together with order and BIT relation has the same expressive power as AC^0. Here, we prove that $\mathcal{NRA}$ is included in AC^0, thus extending half of their results to complex objects.

Proposition 6. *All queries in $\mathcal{NRA}(\leq)$, are in FO-DCL-uniform AC^0 (see [5]).*

We state two more results which help us put the main theorems in perspective. Their proofs are omitted from this extended abstract.

Conservative extension. One may wonder in what sense theorem 4 is a "particular case" of theorem 3. Actually, even though the proof of theorem 4 is quite similar to that of theorem 3, theorem 4 in fact follows from theorem 3, proposition 2 and the conservative extension result presented below.

Paredaens and Van Gucht in [26], and Wong in [33] prove that $\mathcal{NRA}$ is a conservative extension of $\mathcal{NRA}^1$. Suciu in [29] proves that $\mathcal{NRA}(bfix)$ is a conservative extension of $\mathcal{NRA}^1(fix)$, where fix is the usual inflationary fixpoint, and $bfix$ is a bounded version of fix. Using the techniques in [29], we can prove the following:

Proposition 7. *Let Σ be a set of external functions which have set heights ≤ 1. Then, $\mathcal{NRA}(\Sigma, bdcr, \leq)$ is a conservative extension of $\mathcal{NRA}^1(\Sigma, bdcr, \leq)$.*

Note that for the case when $\Sigma = \emptyset$, we can turn the tables and proposition 7 follows directly from the main theorems. For the case when $\Sigma \neq 0$, this proposition requires a separate proof, and we are only able to do it in the presence of order. However, we conjecture that $\mathcal{NRA}(bdcr)$ is a conservative extension of $\mathcal{NRA}^1(dcr)$.

PTIME vs. NC. Immerman, Patnaik and Stemple [22] show that *PTIME* is captured by a language built around *set-reduce* (see section 2). Extending their result also to complex objects we have:

Proposition 8. *$\mathcal{NRA}^1(sri, \leq) = \mathcal{NRA}^1(sri^{(1)}, \leq) = PTIME$ [22] and, for complex objects, $\mathcal{NRA}(bsri, \leq) = \mathcal{NRA}(bsri^{(1)}, \leq) = PTIME$.*

Thus, by the main theorems and this proposition, the difference between *PTIME* and *NC* computable queries over ordered databases can be characterized by the difference between two kinds of recursion on sets. It is interesting to note that only one level of recursion nesting suffices for *sri* and *PTIME*, as opposed to *dcr* and *NC*.

5 Conclusions

Ordering seems to play a crucial role in capturing complexity classes below *NP* and our characterization is no exception. Indeed, it follows from theorem 7.8 in [22] that in the absence of ordering $FO + dcr$ cannot express the lower bound in [9] which is in AC^0 plus parity gates ([9] remark 7.2). As with *PTIME*, *DLOGSPACE*, etc., it remains an important open question whether there exists an r.e. set of "programs" that express exactly the *NC*-computable queries over arbitrary relational databases.

On the other hand, studying the expressiveness of the various forms of recursion on sets in the absence of ordering is quite relevant to query language design. It may also be relevant to complexity theory, if an analog to the surprising result of Abiteboul and Vianu [3] holds. They have shown that *PTIME*$\neq$ *PSPACE* iff first-order least fixpoint queries $\neq$ first-order *while* queries. (Vardi had shown that in the presence of order the *FO*+*while* captures *PSPACE* [32].) Dawar, Lindell, and Weinstein [14] give a machine-independent proof of the Abiteboul and Vianu result making use of properties of bounded variable logics. Abiteboul, Vardi and Vianu [1] give evidence for the robustness of the idea with several such results for other pairs of complexity classes. In our case, the analog would be: $NC \neq PTIME$ iff $FO + dcr \neq FO + sri$ (in our formalism, $\mathcal{NRA}^1(dcr) \neq \mathcal{NRA}^1(sri)$). By setting aside the ordering, with its potential for tricky encodings, this would strengthen the observation (section 4) that the difference between tractable sequential and tractable parallel computation can be characterized as the difference between two ways of recurring on sets.

Acknowledgements. We thank Scott Weinstein for many illuminating discussions, Neil Immerman for answering our sometimes naive queries, Peter Buneman and Leonid Libkin for suggestions from a careful reading of an earlier version of this paper, and Peter, Leonid, and Limsoon Wong for their constant help.

References

1. S. Abiteboul, M. Vardi, and V. Vianu. Fixpoint logics, relational machines, and computational complexity. In *Structure and Complexity*, 1992.
2. Serge Abiteboul and Catriel Beeri. On the power of languages for the manipulation of complex objects. In *Proceedings of International Workshop on Theory and Applications of Nested Relations and Complex Objects*, Darmstadt, 1988. Also available as INRIA Technical Report 846.
3. Serge Abiteboul and Victor Vianu. Generic computation and its complexity. In *Proceedings of 23rd ACM Symposium on the Theory of Computing*, 1991.
4. F. Bancilhon, T. Briggs, S. Khoshafian, and P. Valduriez. FAD, a powerful and simple database language. In *Proceedings of 13th International Conference on Very Large Data Bases*, pages 97–105, 1987.
5. David Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.
6. V. Breazu-Tannen, P. Buneman, and S. Naqvi. Structural recursion as a query language. In *Proceedings of 3rd International Workshop on Database Programming Languages, Naphlion, Greece*, pages 9–19. Morgan Kaufmann, August 1991. Also available as UPenn Technical Report MS-CIS-92-17.
7. V. Breazu-Tannen and R. Subrahmanyam. Logical and computational aspects of programming with Sets/Bags/Lists. In *LNCS 510: Proceedings of 18th International Colloquium on Automata, Languages, and Programming, Madrid, Spain, July 1991*, pages 60–75. Springer Verlag, 1991.
8. Val Breazu-Tannen, Peter Buneman, and Limsoon Wong. Naturally embedded query languages. In J. Biskup and R. Hull, editors, *LNCS 646: Proceedings of 4th International Conference on Database Theory, Berlin, Germany, October, 1992*, pages 140–154. Springer-Verlag, October 1992. Available as UPenn Technical Report MS-CIS-92-47.
9. Jin-Yi Cai, Martin Furer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.
10. Ashok Chandra and David Harel. Computable queries for relational databases. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.
11. P. Clote. Sequential, machine-independent characterizations of the parallel complexity classes $AlogTime$, AC^k, NC^k, and NC. In Samuel R. Buss and Philip J. Scot, editors, *Feasible Mathematics*. Birkhäuser, Boston, 1990.
12. Kevin L. Compton and Claude Laflamme. An algebra and a logic for NC. *Information and Computation*, 87(1/2):240–262, 1990.

13. S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.
14. Anuj Dawar, Steven Lindell, and Scott Weinstein. Infinitary logic and inductive definability over finite structures. *Information and Computation*, 1993. To appear. Available as UPenn Technical Report MS-CIS-91-97.
15. Stephane Grumbach and Victor Vianu. Expressiveness and complexity of restricted languages for complex objects. In *Proceedings of 3rd International Workshop on Database Programming Languages, Naphlion, Greece*, pages 191–202. Morgan Kaufmann, August 1991.
16. Stephane Grumbach and Victor Vianu. Tractable query languages for complex object databases. Technical Report 1573, INRIA, Rocquencourt BP 105, 78153 Le Chesnay, France, December 1991. Extended abstract appeared in PODS 91.
17. Y. Gurevich. Algebra of feasible functions. In *Proceedings of 24th IEEE Symposium on Foundations of Computer Science*, pages 210–214. IEEE Computer Society Press, 1983.
18. Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.
19. Neil Immerman. Expressibility as a complexity measure: Results and directions. In *Proceedings of 2nd Conference on Structure in Complexity Theory*, pages 194–202, 1987.
20. Neil Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987.
21. Neil Immerman. Expressibility and parallel complexity. *SIAM Journal of Computing*, 18:625–638, 1989.
22. Neil Immerman, Sushant Patnaik, and David Stemple. The expressiveness of a family of finite set languages. In *Proceedings of 10th ACM Symposium on Principles of Database Systems*, pages 37–52, 1991.
23. Y. N. Moschovakis. *Elementary Induction on Abstract Structures.* North Holland, 1974.
24. A. Ohori, P. Buneman, and V. Breazu-Tannen. Database programming in Machiavelli, a polymorphic language with static type inference. In James Clifford, Bruce Lindsay, and David Maier, editors, *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 46–57, Portland, Oregon, June 1989.
25. Jan Paredaens and Dirk Van Gucht. Possibilities and limitations of using flat operators in nested algebra expressions. In *Proceedings of 7th ACM Symposium on Principles of Database Systems,Austin, Texas*, pages 29–38, 1988.
26. Jan Paredaens and Dirk Van Gucht. Converting nested relational algebra expressions into flat algebra expressions. *ACM Transaction on Database Systems*, 17(1):65–93, March 1992.
27. D. Stott Parker, Eric Simon, and Patrick Valduriez. SVP: A model capturing sets, streams, and parallelism. In Li-Yan Yuan, editor, *Proceedings of 18th International Conference on Very Large Databases, Vancouver, August 1992*, pages 115–126, San Mateo, California, August 1992. Morgan-Kaufmann.
28. H.-J. Schek and M. H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.
29. Dan Suciu. Fixpoints and bounded fixpoints for complex objects. In Catriel Beeri, Atsushi Ohori, and Dennis Shasha, editors, *Proceedings of 4th International Workshop on Database Programming Languages, New York, August 1993*, pages 263–281. Springer-Verlag, January 1994.

30. Dan Suciu and Val Breazu-Tannen. A query language for NC. In *Proceedings of 13th ACM Symposium on Principles of Database Systems*, pages 167–178, Minneapolis, Minnesota, May 1994.
31. S. J. Thomas and P. C. Fischer. Nested relational structures. In P. C. Kanellakis and F. P. Preparata, editors, *Advances in Computing Research: The Theory of Databases*, pages 269–307, London, England, 1986. JAI Press.
32. M. Y. Vardi. The complexity of relational query languages. In *Proceedings of 14th ACM SIGACT Symposium on the Theory of Computing*, pages 137–146, San Francisco, California, 1982.
33. Limsoon Wong. Normal forms and conservative properties for query languages over collection types. In *Proceedings of 12th ACM Symposium on Principles of Database Systems*, pages 26–36, Washington, D. C., May 1993. See also UPenn Technical Report MS-CIS-92-59.