# Inherent Complexity of Recursive Queries 

View metadata, citation and similar papers at core.ac.uk

E-mail: scosmada@cti.gr

Received January 2000; revised August 13, 2001


#### Abstract

We give lower bounds on the complexity of certain Datalog queries. Our notion of complexity applies to compile-time optimization techniques for Datalog; thus, our results indicate limitations of these techniques. The main new tool is linear first-order formulas, whose depth (respectively, number of variables) matches the sequential (respectively, parallel) complexity of Datalog programs. We define a combinatorial game (a variant of Ehrenfeucht-Fraïssé games) that can be used to prove nonexpressibility by linear formulas. We thus obtain lower bounds for the sequential and parallel complexity of Datalog queries. We prove syntactically tight versions of our results, by exploiting uniformity and invariance properties of Datalog queries. © 2002 Elsevier Science (USA)


## 1. INTRODUCTION

Datalog is a language of negation-free, function-free Horn clauses; it extends the positive existential fragment of relational algebra (the conjunctive queries) by incorporating recursion. Datalog provides a simplified model of the data sublanguage of logic programming [Ul189]. Its computational and expressive power have been the object of considerable research activity.

The recursive facilities of Datalog can be used to express queries such as transitive closure, which cannot be expressed in algebraic query languages based on firstorder logic [AU79]. Attempts to calibrate the expressive power of Datalog by comparing it with other languages include [AG94], which shows that a first-order query is expressible in Datalog iff it is equivalent to an existential query. Specific queries not expressible in Datalog are given in [LM89, KV95] (which give monotonic NP-complete queries) and in [ACY95] (which gives monotonic queries in polynomial time).

In this paper we use tools developed for nonexpressibility, to prove lower bounds on the complexity of Datalog queries. Our motivation comes from compile-time optimization techniques for Datalog, such as magic sets [BMSU86, BR91], counting [BMSU86, SZ88], and the parallelization methods of [AP93, UvG88]. These
techniques transform a program to an equivalent one, which is subsequently evaluated in a standard bottom-up way. Previous work has indicated limitations of such optimizations, by giving lower bounds on various syntactic parameters of a Datalog program [AC89, BKBR90, A97]. Our lower bounds on complexity give more precise information on the scope of these compile-time techniques.

We consider two complexity measures for Datalog programs (Section 2). The derivation tree size is used in [AP93, UvG88]; it captures the parallel complexity of the program. The derivation dag size ${ }^{1}$ captures the sequential space complexity of the program. We give lower bounds for the derivation tree size and the derivation dag size of certain Datalog queries in Sections 4 and 5.

The Datalog queries we consider are variants of the following standard problems:
Path system accessibility: this is a prototypical P-complete query [GJ79]. Our lower bounds imply similar lower bounds for several P-complete Datalog queries [AP93, UvG88].

Same generation: it asks about paths of equal length, in a given directed graph. Variants of this query are among the simplest amenable to magic sets or counting methods [BMSU86].
$K$ node-disjoint paths: $K$ is some fixed integer [KV95]. This problem can be solved in time independent of $K$ using flow techniques. Our lower bounds show that, in contrast, the complexity of Datalog programs for the same problem has to grow exponentially with $K$.

We use a combinatorial game which captures expressibility by first-order formulas [Eh61, Fr54]; we adapt it to the new class of linear formulas, motivated by derivation trees and derivation dags of Datalog programs (Section 3). We use in addition the "pumping" techniques developed in [ACY95, AC89], which can exploit the uniformity of the sequences of formulas obtained from Datalog programs.

In Section 6 we give some open problems suggested by our work.

## 2. PRELIMINARIES

### 2.1. First-Order Queries

A database is a relational structure $\mathscr{D}=\left(D, r_{1}, \ldots, r_{N}\right)$, where each $r_{i}$ is a relation over the domain $D$ (or a constant from $D$ ).

A query is a function with a database as argument, returning a relation of fixed arity $w$. A Boolean query is a query which returns true or false.

The core of algebraic query languages for relational databases is first-order logic over a signature $\left(R_{1}, \ldots, R_{N}\right)$, where each $R_{i}$ is a relation symbol (or constant symbol) denoting the relation (or constant) $r_{i}$ [U1189, AHV95]. The output of the query defined by a formula $\phi$ with free variables $x_{1}, \ldots, x_{w}$ consists of the tuples $\left\langle a_{1}, \ldots, a_{w}\right\rangle$ such that $\phi$ is satisfied (in the database $\mathscr{D}$ ) by assigning the value $a_{i}$ of $D$ to the variable $x_{i}$.

[^0]Definition 1. The transitive closure query Path on a database ( $D, a r c$ ) returns the set of tuples $\langle a, b\rangle$ such that: the directed graph represented by the binary relation $\operatorname{arc}$ has a path from $a$ to $b$.

## Theorem 2. The query Path is not expressible in first-order logic [AU79].

Theorem 2 considers expressibility over finite structures. Instead of a compactness argument (which suffices to prove the result if expressibility is taken over all structures) [AU79] uses quantifier elimination; the technique is developed in full generality in [Gai82]. Theorem 2 can also be shown by the method of EhrenfeuchtFraïssé games [Eh61, Fr54].

Definition 3. In a ( $p, m$ )-Ehrenfeucht-Fraïssé game, Players I and II alternate placing pebbles on the elements of two structures $\mathscr{D}$ and $\mathscr{D}^{\prime}$. Each Player has a set of $p$ pebbles labeled $1, \ldots, p$. If Player I pebbles an element of $\mathscr{D}$ (respectively $\mathscr{D}^{\prime}$ ) with the pebble labeled $i$, Player II has to pebble an element of $\mathscr{D}^{\prime}$ (respectively $\mathscr{D}$ ) with the pebble labeled $i$. Player I is the first to start.

The game is played for $m$ rounds. Player II wins the game if, after each one of his moves, the substructure of $\mathscr{D}$ induced by the pebbled elements is isomorphic to the corresponding substructure of $\mathscr{D}^{\prime}$; where the isomorphism maps the element pebbled by the $i$ th pebble of Player I, to the element pebbled by the $i$ th pebble of Player II.

Theorem 4. Player II has a winning strategy for the ( $p, m$ )-Ehrenfeucht-Fraïssé game on structures $\mathscr{D}, \mathscr{D}^{\prime}$, iff the structures satisfy the same first-order sentences with $p$ variables and quantifier depth ${ }^{2} m$.

Theorem 2 can be shown (using the above result) by finding, for each ( $p, m$ ), finite structures $\mathscr{D}_{p, m}, \mathscr{D}_{p, m}^{\prime}$, such that the query Path returns different results on $\mathscr{D}_{p, m}, \mathscr{D}_{p, m}^{\prime}$; and such that Player II can win the $(p, m)$-game on $\mathscr{D}_{p, m}, \mathscr{D}_{p, m}^{\prime}$.

### 2.2. Datalog Queries

Datalog is a logic programming language, without function symbols and negation [Ull89]. We refer the reader to [Ull89] for the basic definitions; we illustrate them with a simple example.

The query Path can be expressed by the Datalog program:

$$
\begin{align*}
& \operatorname{PATH}(x, y) \leftarrow \operatorname{ARC}(x, y)  \tag{1}\\
& \operatorname{PATH}(x, y) \leftarrow \operatorname{ARC}(x, z), \operatorname{PATH}(z, y) .
\end{align*}
$$

Here the symbol $A R C$ is an EDB-predicate, denoting the database relation arc. The symbol PATH is an IDB-predicate, denoting a relation which is defined by the rules of the program. The relational atomic formulas in a rule of the program are the literals of the rule. We assume that the rules of a Datalog program contain no

[^1]equalities; if they occur, they can be eliminated by appropriate substitutions of variables for their equals.

The relation PATH can be computed by initializing to empty, and repeatedly applying the rules of the program to add tuples to $P A T H$, until no new tuples can be added. The first rule is initialization: if $\left\langle e, e^{\prime}\right\rangle$ is a tuple of arc, the tuple $\left\langle e, e^{\prime}\right\rangle$ is added to $P A T H$. The second rule is recursive: if $\left\langle e, e^{\prime}\right\rangle$ is a tuple of $\operatorname{arc}$ and $\left\langle e^{\prime}, e^{\prime \prime}\right\rangle$ is a tuple of $P A T H$, the tuple $\left\langle e, e^{\prime \prime}\right\rangle$ is added to $P A T H$. It can be seen that a tuple $\langle a, b\rangle$ will be added to $P A T H$ iff the database (considered as a directed graph) contains a path from the element $a$ to the element $b$.

The complexity of a Datalog program can be captured by the notion of derivation tree [UvG88].

For a given database $\mathscr{D}$, a derivation tree has nodes labeled by closed relational atomic formulas (without variables). The vocabulary of these formulas consists of the EDB- and IDB-predicates of the program, and of a set of constant symbols $\{e \mid e$ is a value in the domain of $\mathscr{D}\} .^{3}$

An instantiation of a rule substitutes constant symbols (values in the database domain) for the variables of the rule. This substitution produces, for each literal $\alpha$ of the rule, a closed relational atomic formula, which we call the instantiation of $\alpha$.

Definition 5. For a given Datalog program and database $\mathscr{D}$, a derivation tree is a rooted tree with nodes labeled as follows.
(i) Each leaf is labeled by an atomic formula $R\left(e_{1}, \ldots, e_{w}\right)$, where $R$ is an EDB-predicate and $\left\langle e_{1}, \ldots, e_{w}\right\rangle$ is a tuple in the database relation denoted by $R$.
(ii) Each internal node is labeled by an atomic formula $P\left(e_{1}, \ldots, e_{w}\right)$, where $P$ is an IDB-predicate. The labels of each internal node $u$ and its children come from some instantiation of some rule $\rho$ of the program. Specifically, the instantiation of the literal in the left-hand side of $\rho$ is the label of $u$; and the instantiation of each literal $\alpha$ in the right-hand side of $\rho$ is the label of a child $c_{\alpha}$ of $u$ corresponding to $\alpha$. The edges of the derivation tree are directed, from each child $c_{\alpha}$ of $u$ to its parent; the edge $\left(c_{\alpha}, u\right)$ is labeled with the rule $\rho$ and the literal $\alpha$.

Proposition 6. For a given Datalog program, a database $\mathscr{D}$ and a IDB-predicate $P$, a tuple $\left\langle e_{1}, \ldots, e_{w}\right\rangle$ is in the recursively defined relation $P$ iff there exists a derivation tree with root labeled $P\left(e_{1}, \ldots, e_{w}\right)$.

Thus, a tuple $\langle a, b\rangle$ is in the recursively defined relation $P A T H$ iff there exists a derivation tree with root labeled $\operatorname{PATH}(a, b)$. Note further that, if the domain of the database contains $n$ elements, there exists such a derivation tree with depth $O(n)$, and size $O(n)$. A comparison may be made with the equivalent Datalog program below, which computes the same relation $P A T H$.

$$
\begin{align*}
& \operatorname{PATH}(x, y) \leftarrow \operatorname{ARC}(x, y)  \tag{2}\\
& \operatorname{PATH}(x, y) \leftarrow \operatorname{PATH}(x, z), \operatorname{PATH}(z, y)
\end{align*}
$$

${ }^{3} \mathrm{We}$ are abusing notation here, for the sake of brevity.

It can be seen that, for program 2, there exists a derivation tree with depth $O(\log n)$; and size $O(n)$.
If the depth of derivation trees for a given Datalog program is (at most) $d(n)$ (for databases of size $n$ ), the program can be evaluated with polynomially many proces-sors-by executing all possible applications of rules in parallel-in parallel time $O(d(n) \log n)$ [UvG88]. Optimization methods presented in [AP93, UvG88] transform (at compile time) a Datalog program with derivation tree size $s(n)$, to an equivalent program with derivation tree depth $O(\log s(n))$; and thus with parallel time complexity $O(\log s(n) \log n)$.

In Sections 4 and 5 we give lower bounds for the derivation tree size of certain Datalog queries. Our results delimit the scope of compile time techniques as in [AP93, UvG88].
The sequential complexity of a Datalog program can be measured by the number of intermediate tuples (of the recursively defined relations) which must be produced, before a given tuple appears in the result. Compile time techniques such as magic sets and counting [BMSU86, BR91, SZ88] transform a Datalog program to an equivalent program which produces fewer intermediate tuples.

For a given result tuple, intermediate tuples appear as labels of the internal nodes of a derivation tree. In a derivation tree several nodes may have the same label. To obtain a more accurate representation we consider a derivation dag, which is a graph obtained from a derivation tree by identifying nodes with the same label.

Defintion 7. A derivation tree is minimal if, whenever a node $u$ is an ancestor of a node $v$, the nodes $u, v$ have different labels.

It is not hard to transform a given derivation tree to a smaller one which is minimal [UvG88].

Defintion 8. A derivation dag is obtained from a minimal derivation tree $\tau$ as follows.

We identify nodes of $\tau$ with the same label. The graph $\delta$ obtained is a dag (by minimality of $\tau$ ), with sources the leaves of $\tau$, and a unique sink which is the root of $\tau$. The directed edges of $\tau$ become arcs of $\delta$.

If the arcs coming into a node of $\delta$ correspond to several instantiations of rules of the program, only the arcs corresponding to one instantiation (of some rule) are kept.

Note that a derivation dag can be exponentially smaller than the original derivation tree.

In Sections 4 and 5 we give lower bounds for the number of intermediate tuples of certain Datalog queries. In some cases, the bounds are obtained from lower bounds for the derivation dag size. Our results delimit the scope of compile time techniques as in [BMSU86, BR91, SZ88].

To cover programs obtained from counting methods, we assume that a separate domain Int of integers is available. Specifically, a database is a structure $\mathscr{D}=\left(D\right.$, Int $\left., r_{1}, \ldots, r_{N}, 0, \lambda x . x+1\right)$, where each $r_{i}$ is a relation over the domain $D$ (or a constant from $D$ ). The additional symbols $Z E R O$ (an integer constant) and INCREMENT (an integer function of one argument) can be used in Datalog
programs. We call programs as above-produced by counting methods-ExtendedDatalog programs. The EDB-predicates of an Extended-Datalog program denote relations over the domain $D$, as in standard Datalog. The IDB-predicates of an Extended-Datalog program denote relations over $D \cup I n t$.

### 2.3. Formulas and Games for Datalog

A query defined by a Datalog program can by expressed by an infinite sequence of existential positive first-order formulas. ${ }^{4}$ We say that a query $Q$ is expressed by a sequence of formulas with $v(n)$ variables and $q(n)$ quantifier depth, if, for every database $\mathscr{D}$ of size $n$ and every tuple $\left\langle a_{1}, \ldots, a_{w}\right\rangle$ of elements of $D$ :
$\left\langle a_{1}, \ldots, a_{w}\right\rangle \in Q(\mathscr{D})$ if there is a formula $\phi$ in the sequence, such that $\phi$ is satisfied in $\mathscr{D}$ by the values $a_{i}$.
$\left\langle a_{1}, \ldots, a_{w}\right\rangle \in Q(\mathscr{D})$ only if there is a formula $\phi$ in the sequence with at most $v(n)$ variables and at most $q(n)$ quantifier depth, such that $\phi$ is satisfied in $\mathscr{D}$ by the values $a_{i}$.

A sequence of formulas expressing $P A T H$ can be obtained-from the example program 1-by iterating the operator

$$
\begin{aligned}
\Theta(\phi) \equiv & \phi(x, y) \vee A R C(x, y) \\
& \vee \exists z \cdot(A R C(x, z) \wedge \phi(z, y))
\end{aligned}
$$

using as starting point the formula false (defining the empty relation). The formula obtained after $n$ iterations expresses the relation PATH on databases of size $n$.

To express a Datalog query on databases of size $n$ we need $d(n)$ iterations of the operator $\Theta$ obtained from the program-where $d(n)$ is the derivation tree depth of the program.

The following nontrivial refinement of the above observation is shown in [KV95, LM89]:

Theorem 9. A query defined by a Datalog program can be expressed by a sequence of existential positive formulas with a constant ${ }^{5}$ number of variables; and quantifier depth bounded by the derivation tree depth of the program.

Expressibility by existential positive formulas can be captured by existential positive Ehrenfeucht-Fraïssé games [KV95, LM89]; they modify ordinary EhrenfeuchtFraïssé games as follows:
(i) Player I always plays on the same structure $\mathscr{D}$, and Player II responds on $\mathscr{D}^{\prime}$.
(ii) Player II wins the game if, after each one of his moves, the substructure of $\mathscr{D}^{\prime}$ induced by his pebbles is a homomorphic image of the substructure of $\mathscr{D}$ induced by the pebbles of Player I; where the homomorphism maps the element

[^2]pebbled by the $i$ th pebble of Player I, to the element pebbled by the $i$ th pebble of Player II.

We recall the definition of a homomorphism between relational structures. Let $\mathscr{D}=\left(D, r_{1}, \ldots, r_{N}\right), \mathscr{D}^{\prime}=\left(D^{\prime}, r_{1}^{\prime}, \ldots, r_{N}^{\prime}\right)$ be structures with the same signature.

A function $h$ from the domain $D$ to the domain $D^{\prime}$ is a homomorphism from $\mathscr{D}$ to $\mathscr{D}^{\prime}$, if for each tuple $\left\langle a_{1}, \ldots, a_{w_{i}}\right\rangle$ of $r_{i},\left\langle h\left(a_{1}\right), \ldots, h\left(a_{w_{i}}\right)\right\rangle$ is a tuple of $r_{i}^{\prime}$.

Theorem 4 has the following analogue [KV95, LM89]:
Theorem 10. Player II has a winning strategy for the ( $p, m$ )-existential positive Ehrenfeucht-Fraïssé game on structures $\mathscr{D}$, $\mathscr{D}^{\prime}$ iff: every existential positive first-order sentence with $p$ variables and quantifier depth $m$ which is true in $\mathscr{D}$, is also true in $\mathscr{D}^{\prime}$.
$\operatorname{Datalog}(\neq)$ is an extension of Datalog which allows literals of the form $t \neq t^{\prime}$ in bodies of rules, where $t, t^{\prime}$ are variables or constants. A result analogous to Theorem 9 holds for $\operatorname{Datalog}(\neq)$ and existential positive formulas which allow in addition negations of equalities [KV95].

Expressibility by existential positive formulas with $\neq$ can be captured by existential positive Ehrenfeucht-Fraïssé games with $\neq$ [KV95]; they modify ordinary Ehrenfeucht-Fraïssé games as follows:
(i) Player I always plays on the same structure $\mathscr{D}$, and Player II responds on $\mathscr{D}^{\prime}$.
(ii) Player II wins the game if, after each one of his moves, the substructure of $\mathscr{D}^{\prime}$ induced by his pebbles is a homomorphic image of the substructure of $\mathscr{D}$ induced by the pebbles of Player I; where the homomorphism maps the element pebbled by the $i$ th pebble of Player I, to the element pebbled by the $i$ th pebble of Player II; and the homomorphism is one-to-one.

A result analogous to Theorem 10 holds for existential positive EhrenfeuchtFraïssé games with $\neq$ and existential positive sentences with $\neq$ [KV95].

All of the above results can be adapted straightforwardly to Extended-Datalog.

## 3. LINEAR FORMULAS AND LINEAR GAMES

In this section we define the linear first-order formulas, which specialize the existential positive formulas. The syntactic parameters of linear formulas (depth, number of variables) match more closely the complexity measures we are interested in (derivation dag size and derivation tree size of Datalog programs). We define a special kind of Ehrenfeucht-Fraïssé games, whose parameters (number of moves and number of pebbles) match the syntactic parameters of linear formulas.

Let $\bar{x}$ be a (possibly empty) sequence of variables, $x_{1} \cdots x_{s}$. We denote by $\exists \bar{x}$ the sequence of existential quantifications $\exists x_{1} \cdots \exists x_{s}$.

Definition 11. Let $\beta_{k}, k=1, \ldots, m$, be a sequence of quantifier-free formulas without negation; also, $\beta_{k}$ is allowed to be the formula true. Denote by $W$ the maximum of the number of variables of the $\beta_{k}$ 's.

Let $\phi_{k}, k=1, \ldots, m$ be a sequence of formulas as follows:

$$
\begin{aligned}
\phi_{1} & \equiv \beta_{1} \\
\phi_{k} & \equiv \beta_{k} \wedge \exists \bar{x}_{k} \cdot \phi_{k-1}
\end{aligned}
$$

where $\bar{x}_{k}$ is a (possibly empty) sequence of variables, for $k=2, \ldots, m$.
We call $\phi_{m}$ a linear formula of depth $m$ and width $W$.
If $\beta_{k}, k=1, \ldots, m$, is a sequence of quantifier free formulas without negations of relations, but with negations of equalities, we call $\phi_{m}$ a linear formula with $\neq$, of depth $m$ and width $W$.

Note that the depth and the width of a linear formula depend on the specific choice of the sequence $\beta_{k}$.

It turns out that Datalog queries can be defined by linear formulas of constant width. The following results bound the depth and number of variables of linear formulas, in terms of the derivation dag size and the derivation tree size of the program.

Theorem 12. A query $Q$ defined by a Datalog program (with $\neq$ ) with derivation dag size $S(n)$ can be expressed by a sequence of linear formulas (with $\neq$ ) with depth $S(n)+1$, and width depending only on the program.

Proof. Let $\mathscr{D}$ be a database of size $n$, and let $\left\langle a_{1}, \ldots, a_{w}\right\rangle$ be a tuple in $Q(\mathscr{D})$. Let $\delta$ be a derivation dag with sink $g$ labeled $P\left(a_{1}, \ldots, a_{w}\right)$, where $P$ is the output IDBpredicate of the program. The dag $\delta$ has $m$ nodes, where $m \leqslant S(n)$.

We construct from $\delta$ a linear formula $\phi$. For each constant $e$ occurring in (a label of) the derivation lag, the formula uses a variable $x_{e}$. The free variables of $\phi$ are $x_{a_{1}}, \ldots, x_{a_{w}}$.

The formula will be satisfied in $\mathscr{D}$ iff: there exists a derivation dag $\Delta$ such that $\delta$ is a homomorphic image of $\Delta .{ }^{6}$ The depth of $\phi$ will be $m+1$.

We number the nodes of $\delta$ according to some topological sort. For $k=1, \ldots, m$, let $f^{k}$ be the $k$ th node in the numbering, and let $\left(f_{1}^{k}, f^{k}\right), \ldots,\left(f_{s}^{k}, f^{k}\right)$ be the arcs coming into $f^{k}$. Note that each $f_{i}^{k}$ comes before $f^{k}$ in the topological sort. Recall that each of the arcs coming into $f^{k}$ is labelled by a rule $\rho$ of the program; and that the labels of the nodes $f_{1}^{k}, \ldots, f_{s}^{k}, f^{k}$ are atomic formulas, obtained by substituting constants into the literals of $\rho$, according to some instantiation I.

Let $\beta_{k}$ be a quantifier-free formula constructed as follows: we start by forming a conjunction of all the literals occurring in the rule $\rho$. We then substitute, for each variable $z$, the variable $x_{e}$, where $e$ is the value assigned to $z$ by the instantiation I. Observe that $\beta_{k}$ has no disjunctions or equalities. Note also that the maximum number of variables of $\beta_{k}$ is the same as the number of variables of the rule $\rho$; therefore, it depends only on the program.

We put

$$
\begin{aligned}
\phi_{1} & \equiv \beta_{1} \\
\phi_{k} & \equiv \beta_{k} \wedge \phi_{k-1} \quad \text { for } \quad k=2, \ldots, m
\end{aligned}
$$

${ }^{6}$ I.e., $\delta$ is obtained from $\Delta$-up to isomorphism-by equating some values.

The desired formula is $\exists \bar{x} \cdot \phi_{m}$, where $\bar{x}$ contains all the variables of $\phi_{m}$ except $x_{a_{1}}, \ldots, x_{a_{w}}$.

Remark 13. In the case of a Boolean query, the proof of the above result produces a sequence of sentences of the form $\exists v_{1} \cdots \exists v_{p} . \phi$-where $\phi$ is a linear formula (with $\neq$ ) with variables $v_{1}, \ldots, v_{p}$.

Lemma 14. Let $\phi, \phi^{\prime}$ be linear formulas (with $\neq$ ) of width $W$. Let $V, V^{\prime}$ be the set of variables occurring in $\phi, \phi^{\prime}$ respectively, and assume that the free variables of $\phi^{\prime}$ are not bound in $\phi$.

The formula $\phi \wedge \phi^{\prime}$ is equivalent to a linear formula (with $\neq$ ) of width $W$, with variables $V \cup V^{\prime}$.

Proof. Let $\phi$ be $\phi_{m}$ as in Definition 11, where for $k=1, \ldots, m, \phi_{k} \equiv$ $\beta_{k} \wedge \exists \bar{x}_{k} . \phi_{k-1}$.

Replace the subformula $\beta_{1}$ of $\phi$ with the formula $\beta_{1} \wedge \phi^{\prime}$. The result is the desired linear formula.

We denote a formula with free variables $x_{1}, \ldots, x_{w}$ by $\phi\left(x_{1}, \ldots, x_{w}\right)$, or $\phi(\bar{x})$.
Theorem 15. A query $Q$ defined by a Datalog program (with $\neq$ ) with derivation tree size $s(n)$ can be expressed by a sequence of linear formulas (with $\neq$ ) with $O(\log s(n))$ variables, and width depending only on the program.

Proof. Let $\mathscr{D}$ be a database of size $n$, and let $\left\langle a_{1}, \ldots, a_{w}\right\rangle$ be a tuple in $Q(\mathscr{D})$. Let $\tau$ be a derivation tree with root $u$ labeled $P\left(a_{1}, \ldots, a_{w}\right)$, where $P$ is the output IDBpredicate of the program. The tree $\tau$ has at most $s(n)$ nodes.

We construct from $\tau$ a linear formula $\phi$. For each constant $e$ occurring in (a label of) the derivation tree, the formula uses a variable $x_{e}$. The free variables of $\phi$ are $x_{a_{1}}, \ldots, x_{a_{w}}$.

The formula is satisfied in $\mathscr{D}$ iff: there exists a derivation tree $T$ such that $\tau$ is a homomorphic image of $T .^{7}$

Let $c_{1}, \ldots, c_{s}$ be the children of the root $u$ of $\tau$. Recall that each of the arcs coming into $u$ (from one of its children) is labelled by a rule $\rho$ of the program; and that the labels of the nodes $c_{1}, \ldots, c_{s}, u$ are atomic formulas, obtained by substituting constants into the literals of $\rho$, according to some instantiation I.

We construct the formula $\phi$ by induction on the depth of the derivation tree. For each subtree $\tau_{i}$ of $\tau$ with root $c_{i}$, we construct a formula $\phi_{i}\left(\bar{x}_{i}\right)$. The sets of variables $\bar{x}_{i}, i=1, \ldots, s$ are pairwise disjoint. Also, the variables in $\bar{x}_{1}$ through $\bar{x}_{i-1}$ are not bound in $\phi_{i}$.

Let $\beta$ be a quantifier-free formula constructed as follows: we start by forming a conjunction of all the literals occurring in the rule $\rho$. We then substitute, for each variable $z$, the variable $x_{e}$, where $e$ is the value assigned to $z$ by the instantiation I. Observe that $\beta$ has no disjunctions or equalities. Note also that the maximum number of variables of $\beta$ is the same as the number of variables of the rule $\rho$; therefore, it depends only on the program.

[^3]We put

$$
\phi \equiv \exists \bar{x} .\left[\beta \wedge \phi_{s} \wedge\left(\cdots\left(\phi \wedge \phi_{1}\right) \cdots\right)\right],
$$

where $\bar{x}$ contains all the variables of the $\bar{x}_{i}$ 's except $x_{a_{1}}, \ldots, x_{a_{n}}$.
By Lemma 14, the formula $\phi$ is equivalent to a linear formula with the same variables.

The variables of $\phi_{1}$ are re-used in each $\phi_{i}$, with the proviso that the variables in $\bar{x}_{1}$ through $\bar{x}_{i-1}$ are not bound in $\phi_{i}$.

Suppose that $M$ variables are available. Let $T_{M}$ be a derivation tree, of least size, with the following property: the formula obtained as above from $T_{M}$ has to use all of $M$ variables.

Note that $T_{M}$ may not exist for some $M$. For example, if every rule of the Datalog program has at most one subgoal (i.e., the program is linear [UvG88, Ull89]) the derivation tree $T_{M}$ only exists for bounded $M$. If $T_{M}$ does not exist, we take its size to be infinite.

We claim that there is a constant $K$, depending only on the program, such that the following hold for $M>K$ :
(i) the root $u$ of $T_{M}$ has at least two children;
(ii) let $c_{1}, \ldots, c_{s}$ be the children of $u$, and let $T_{i}$ be the subtree of $T_{M}$ rooted at $c_{i}$; then $T_{i}$ is $T_{M_{i}}$ for some $M_{i}>M-K$;
(iii) every path of $T_{M}$ from the root to a leaf has length at least $M / K$.

The claim can be shown by a simple simultaneous induction on the depth of $T_{M}$; we use the minimality of $T_{M}$, as well as the construction of a formula from $T_{M}$. If for some $M$ no $T_{M}$ exists, we take the claim to be vacuously true.

It follows that the size of $T_{M}$ is at least exponential in $M$. Thus, the formula obtained from a derivation tree of size $s(n)$ has $O(\log s(n))$ variables.

Theorem 12 and Theorem 15 can be adapted straightforwardly to ExtendedDatalog.
We now show that expressibility by linear formulas can be captured by a modification to the existential positive Ehrenfeucht-Fraissé game.

Defintion 16. The ( $p, m, W$ )-linear Ehrenfeucht-Fraïsé game is played between Players I and II, on two structures $\mathscr{D}$ and $\mathscr{D}^{\prime}$. Each Player has a set of $p$ pebbles, labeled $1, \ldots, p$. At each point in the game some of Player I's pebbles (respectively Player II's) are on elements of $\mathscr{D}$ (respectively $\mathscr{D}^{\prime}$ ).

Just before the game starts there are no pebbles on either structure.
A round of the game consists of a move of Player I , and a corresponding move of Player II.

A move of Player I consists of picking up some of his pebbles and placing them on elements of $\mathscr{D}$. These pebbles may already have been placed on $\mathscr{D}$ at previous moves - in that case they are moved to new positions. Subsequently, Player I chooses a set of at most $W$ of his pebbles; he designates this set as the window of the round.

A move of Player II, corresponding to a given move of Player I, consists of placing some of Player II's pebbles to elements of $\mathscr{D}^{\prime}$. Specifically, Player II places on $\mathscr{D}^{\prime}$ the pebbles with the same labels as the pebbles Player I placed on $\mathscr{D}$.

Player I starts the game by playing a sequence of $m$ moves. Player II then responds by playing a sequence of $m$ moves, in one-to-one correspondence with the moves of Player I. Thus, a sequence of $m$ rounds is formed.

Player II wins if, after each one of his moves: there is a homomorphism from the substructure of $\mathscr{D}$ induced by those pebbles of Player I that were designated as the window of that round, to the substructure of $\mathscr{D}^{\prime}$ induced by Player II's pebbles; moreover, this homomorphism has to map the element pebbled by the $i$ th pebble of Player I, to the element pebbled by the $i$ th pebble of Player II.

The $(p, m, W)$-linear Ehrenfeucht-Fraïssé game with $\neq$ is defined in the same way: we additionally require that the above homomorphism is one-to-one.

Theorem 17. (i) Player II has a winning strategy for the ( $p, m, W$ )-linear Ehrenfeucht-Fraïssé game on structures $\mathscr{D}, \mathscr{D}^{\prime}$ iff: every sentence $\exists v_{1} \cdots \exists v_{p} . \phi$-where $\phi$ is a linear formula with variables $v_{1}, \ldots, v_{p}$, depth $m$ and width $W$-that is true in $\mathscr{D}$, is also true in $\mathscr{D}^{\prime}$.
(ii) Player II has a winning strategy for the ( $p, m, W$ )-linear EhrenfeuchtFraïssé game with $\neq$ on structures $\mathscr{D}, \mathscr{D}^{\prime}$ iff: every sentence $\exists v_{1} \cdots \exists v_{p} . \phi$-where $\phi$ is a linear formula with $\neq$ with variables $v_{1}, \ldots, v_{p}$, depth $m$ and width $W$-that is true in $\mathscr{D}$, is also true in $\mathscr{D}^{\prime}$.

## Proof. If:

From a sequence of $m$ moves of Player I, we construct a sequence of formulas $\phi_{k}$, $k=1, \ldots, m$ as in Definition 11. The formula $\phi_{k}$ describes the windows of rounds 1 through $k$.

We use variables $v_{1}, \ldots, v_{p}$; the variable $v_{i}$ corresponds to the pebble labeled $i$.
In the $k$ th move Player I places some pebbles on elements of $\mathscr{D}$. He then designates pebbles $i_{1}, \ldots, i_{W}$ as the window of the $k$ th round; at that point, these pebbles are on $\mathscr{D}$, on the elements $a_{1}, \ldots, a_{W}$ respectively.

For (i), let $\beta_{k}$ be a quantifier-free formula with variables $v_{i_{1}}, \ldots, v_{i_{W}}$, which is the conjunction of the atomic formulas (relations and equalities) which hold in $\mathscr{D}$ by assigning to $v_{i_{1}}, \ldots, v_{i_{W}}$ the value $a_{1}, \ldots, a_{W}$ respectively.

For (ii), let $\beta_{k}$ be a quantifier-free formula with variables $v_{i_{1}}, \ldots, v_{i_{W}}$, which is the conjunction of the atomic formulas (relations and equalities) and of the negations of equalities which hold in $\mathscr{D}$ by assigning to $v_{i_{1}}, \ldots, v_{i_{W}}$ the value $a_{1}, \ldots, a_{W}$ respectively.

We put

$$
\phi_{k} \equiv \beta_{k} \wedge \exists \bar{x}_{k} \cdot \phi_{k-1}
$$

where $\bar{x}_{k}$ are the variables corresponding to the pebbles Player I placed on $\mathscr{D}$ in the $k$ th move, that were already on $\mathscr{D}$. In particular, $\phi_{1} \equiv \beta_{1}$.

Consider the sentence $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$. This sentence is true in $\mathscr{D}$-this can be seen by assigning to each occurrence of $v_{i_{1}}, \ldots, v_{i_{W}}$ in $\beta_{k}$ the value $a_{1}, \ldots, a_{W}$ respectively.

Thus the sentence $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$ is true in $\mathscr{D}^{\prime}$. We now construct a sequence of $m$ moves of Player II.

Recall that $\phi_{k} \equiv \beta_{k} \wedge \exists \bar{x}_{k} \cdot \phi_{k-1}$. We examine each variable $v_{i}, i=1, \ldots, p$.
If the variable $v_{i}$ occurs in the sequence $\bar{x}_{k}$, let $k^{\prime}$ be the least index such that $k^{\prime} \geqslant k$ and $v_{i}$ occurs in $\beta_{k^{\prime}}$; let $e_{i}^{\prime}$ be the element of $\mathscr{D}^{\prime}$ that was assigned to the occurences of $v_{i}$ in $\beta_{k^{\prime}}$ (to satisfy the sentence $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$ ).

If the variable $v_{i}$ occurs in $\beta_{k}$ (and does not occur in the sequence $\bar{x}_{k}$ ), let $e_{i}^{\prime}$ be the element of $\mathscr{D}^{\prime}$ that was assigned to the occurences of $v_{i}$ in $\beta_{k}$ (to satisfy the sentence $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$ ).

In his $k$ th move Player II picks up every pebble labeled $i$ for which the element $e_{i}^{\prime}$ has been defined above; and places it on the element $e_{i}^{\prime}$ of $\mathscr{D}^{\prime}$.

A simple induction on $k$ shows that Player II wins the game.
Only if:
Let $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$ be a sentence (for (ii), with $\neq$ ) that is true in $\mathscr{D}$. As in Definition 11, we have

$$
\phi_{k} \equiv \beta_{k} \wedge \exists \bar{x}_{k} \cdot \phi_{k-1}
$$

for $k=1, \ldots, m$, where $\beta_{k}$ has at most $W$ variables.
We construct a sequence of $m$ moves of Player I as follows: in his $k$ th move, Player I examines each variable $v_{i}, i=1, \ldots, p$.

If the variable $v_{i}$ occurs in the sequence $\bar{x}_{k}$, let $k^{\prime}$ be the least index such that $k^{\prime} \geqslant k$ and $v_{i}$ occurs in $\beta_{k^{\prime}}$; let $e_{i}$ be the element of $\mathscr{D}$ that was assigned to the occurences of $v_{i}$ in $\beta_{k^{\prime}}$ (to satisfy the sentence $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$ ).

If the variable $v_{i}$ occurs in $\beta_{k}$ (and does not occur in the sequence $\bar{x}_{k}$ ), let $e_{i}$ be the element of $\mathscr{D}$ that was assigned to the occurences of $v_{i}$ in $\beta_{k}$ (to satisfy the sentence $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$ ).

In his $k$ th move Player I picks up every pebble labeled $i$ for which the element $e_{i}$ has been defined above; and places it on the element $e_{i}$ of $\mathscr{D}$. He then designates pebbles $i_{1}, \ldots, i_{W}$ as the window of the $k$ th round, where $v_{i_{1}}, \ldots, v_{i_{W}}$ are the variables occurring in $\beta_{k}$.

Player II wins the game (for (ii), the game with $\neq$ ) by playing a sequence of $m$ corresponding moves.

After the $k$ th move of Player II, the pebble labeled $i$ is lying on the element $e_{i}^{\prime}$ of $\mathscr{D}^{\prime}$. A simple induction on $k$ shows that: $\phi_{k}$ becomes true in $\mathscr{D}^{\prime}$, by assigning to each occurrence of $v_{i_{1}}, \ldots, v_{i_{W}}$ in $\beta_{k}$ the value $e_{i_{1}}^{\prime}, \ldots, e_{i_{W}}^{\prime}$ respectively.

Thus, the sentence $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$ is true in $\mathscr{D}^{\prime}$.
We also have the following weak version of the ( $p, m, W$ )-linear EhrenfeuchtFraïssé game with $\neq$ :

Players I and II play on a structure $\mathscr{D}$. Player I starts the game by playing a sequence of $m$ moves on $\mathscr{D}$. Player II responds by picking $a$ structure $\mathscr{D}^{\prime}$; and playing a sequence of $m$ corresponding moves on $\mathscr{D}^{\prime}$. The winning condition is the same as in the ordinary game with $\neq$.

Arguing as in the Proof of Theorem 17, we show:
If Player II has a winning strategy for the weak version of the ( $p, m, W$ )-linear Ehrenfeucht-Fraïssé game with $\neq$ on structure $\mathscr{D}$ then: for every sentence $\exists v_{1} \cdots \exists v_{p} . \phi$-where $\phi$ is a linear formula with $\neq$ with variables $v_{1}, \ldots, v_{p}$, depth $m$ and width $W$-that is true in $\mathscr{D}$, there is a structure $\mathscr{D}^{\prime}$ such that the sentence is also true in $\mathscr{D}^{\prime}$.
(To prove this, we construct from the sentence $\exists v_{1} \cdots \exists v_{p} . \phi$ a sequence of moves of Player I; we consider the structure $\mathscr{D}^{\prime}$ picked by Player II, and the sequence of corresponding moves of Player II on $\mathscr{D}^{\prime}$.)

Theorem 17 can be adapted straightforwardly to Extended-Datalog.

## 4. LOWER BOUNDS FOR DATALOG QUERIES

### 4.1. Lower Bounds for Derivation Dag Size

The following class of Datalog programs has been singled out in the study of optimization methods [U1189, UvG88].

Definition 18. (i) A Datalog program is piecewise linear if the body of every rule contains at most one IDB-predicate which is mutually recursive with the IDBpredicate in the head of the rule.
(ii) A Datalog program is linear if the body of every rule contains at most one IDB-predicate.

The derivation trees of piecewise linear Datalog programs are "skinny." It can be shown that their size is related polynomially to their depth. Consequently, piecewise linear Datalog programs can be evaluated by efficient parallel algorithms [UvG88]. It has been shown that certain Datalog queries that can be evaluated by efficient parallel algorithms, cannot be defined by linear Datalog programs [AC89].

For linear Datalog programs the size of derivation trees is related linearly to the depth; and coincides with the size of derivation dags.

In [KV95] it is shown that the following query can be expressed by a piecewise linear Datalog $(\neq)$ program.

Definition 19. Let $K$ be a fixed integer. The $K$ disjoint paths query $K$-Paths on a database ( $D, \operatorname{arc}$ ) returns the set of tuples $\langle a, b\rangle$ such that: the directed graph represented by the binary relation arc has $K$ node-disjoint paths from $a$ to $b$.

The piecewise linear Datalog program which expresses $K$-Paths has derivation dag size $O\left(n^{K}\right)$.

We will show the following lower bound:
Theorem 20. Every piecewise linear Datalog $(\neq)$ program that expresses $K$-Paths has derivation dag size $\Omega\left(n^{K}\right)$.

We use the following observation: ${ }^{8}$

[^4]Theorem 21. Every piecewise linear Datalog program can be effectively transformed into an equivalent linear program.

We illustrate the proof of Theorem 21 by an example. We use the following prototypical P-complete [GJ79] query.

Definition 22. The path system accessibility query Access on a database ( $D$, reach, source) is expressed by the following Datalog program:

$$
\begin{align*}
\operatorname{ACCESS}(x) \leftarrow & \operatorname{SOU} R C E(x) \\
\operatorname{ACCESS}(x) \leftarrow & R E A C H(x, y, z)  \tag{3}\\
& A C C E S S(y), \operatorname{ACCESS}(z)
\end{align*}
$$

Access cannot be defined by a piecewise linear Datalog program-see the remarks in the last section of [AC89]. The following piecewise linear variants of Access provide hard cases of Theorem 21.

Definition 23. For $k \geqslant 0$, the query $A c c e s s_{k}$ on a database ( $D$, reach, source) is expressed by the following Datalog program:

$$
\begin{align*}
\operatorname{ACCESS}_{k}(x) \leftarrow & \operatorname{SOURCE}(x) \\
\operatorname{ACCESS}_{k}(x) \leftarrow & R E A C H(x, y, z), \\
& A C C E S S_{k}(y), \operatorname{ACCESS}_{k-1}(z) \\
\operatorname{ACCESS}_{0}(x) \leftarrow & \operatorname{SOURCE}(x)  \tag{4}\\
\operatorname{ACCESS}_{0}(x) \leftarrow & R E A C H(x, y, z), \\
& A C C E S S_{0}(y), \operatorname{SOURCE}(z)
\end{align*}
$$

We show below a linear Datalog program which defines Access $_{1}$. The program uses an IDB-predicate $Q_{1}\left(x, x^{\prime}\right)$ which simulates the conjunction of $\operatorname{ACCESS} S_{1}(x)$, $A C C E S S_{0}\left(x^{\prime}\right)$ :

$$
\begin{align*}
& \operatorname{ACCESS}_{1}(x) \leftarrow \operatorname{SOURCE}(x) \\
& \operatorname{ACCESS}_{1}(x) \leftarrow \operatorname{REACH}(x, y, z), Q_{1}(y, z) \\
& Q_{1}\left(x, x^{\prime}\right) \leftarrow \operatorname{ACCESS_{1}(x),\operatorname {SOURCE}(x^{\prime })}  \tag{5}\\
& Q_{1}\left(x, x^{\prime}\right) \leftarrow \leftarrow \operatorname{REACH}\left(x^{\prime}, y, z\right), \\
& Q_{1}(x, y), \operatorname{SOURCE}(z)
\end{align*}
$$

The first rule for $Q_{1}\left(x, x^{\prime}\right)$ handles the initialization rule for $\operatorname{ACCESS} S_{0}\left(x^{\prime}\right)$. The second rule for $Q_{1}\left(x, x^{\prime}\right)$ handles the recursive rule for $\operatorname{ACCESS} S_{0}\left(x^{\prime}\right)$. Observe that the variable $x$ appears in both sides of the second rule-because of the postponement of $A C C E S S_{1}(x)$. Such variables are called persistent [A97, AF99].

By iterating the above construction we obtain a linear program defining Access $_{k}$. The linear program for Access $_{2}$ will use an IDB-predicate $Q_{2}\left(x, x^{\prime}, x^{\prime \prime}\right)$ simulating
the conjunction of $\operatorname{ACCESS}_{2}(x), Q_{1}\left(x^{\prime}, x^{\prime \prime}\right)$. The linear program thus obtained for Access $_{k}$ will contain a rule with $k$ variables appearing in both sides.

Theorem 20 follows from Theorem 21 and the following result:
Theorem 24. Every linear Datalog $(\neq)$ program that expresses $K$-Paths has derivation dag size $\Omega\left(n^{K}\right)$.

Proof. We consider a Boolean version of $K$-Paths. The query $K$-Paths $s_{\text {bool }}$ on a database ( $D$, arc, $a, b$ )-where $a, b$ are constant symbols-returns true iff: the directed graph represented by the binary relation arc has $K$ node-disjoint paths from $a$ to $b$. It suffices to show that every linear $\operatorname{Datalog}(\neq)$ program that expresses $K$-Path $s_{\text {bool }}$ has derivation dag size $\Omega\left(n^{K}\right)$.

Suppose $\Pi$ is a linear $\operatorname{Datalog}(\neq)$ program that expresses $K-$ Path $_{\text {bool }}$, with derivation dag size $S(n)$. By Theorem 12, $K$ - Path $_{s_{\text {bool }}}$ can be expressed by a sequence of sentences of the form $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$-where $\phi$ is a linear formula with $\neq$ with variables $v_{1}, \ldots, v_{p}$, depth $S(n)+1$, and width depending only on $\Pi$. Moreover we have, as in the Proof of Theorem 12,

$$
\phi_{k} \equiv \beta_{k} \wedge \phi_{k-1}
$$

for $k=1, \ldots, m$. Recall that $\beta_{k}$ does not contain equalities.
By the linearity of the $\operatorname{Datalog}(\neq)$ program, every variable of $\beta_{k}$ which occurs in $\phi_{k-1}$, occurs in $\beta_{k-1}$.

We will show that, if the depth of $\phi_{m}$ is at most $\gamma n^{K}$-where $\gamma$ is an appropriately chosen constant depending only on $\Pi$-we can construct a structure $\mathscr{D}^{\prime}$ such that: the sentence $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$ is true in $\mathscr{D}^{\prime}$, but the query $K$ - Paths $s_{\text {bool }}$ returns false on $\mathscr{D}^{\prime}$. It follows that $S(n)$ is $\Omega\left(n^{K}\right)$.

Consider a structure $\mathscr{D}$ of size $n$, consisting of $K$ node-disjoint paths of equal length between two nodes $a, b$. The query $K$-Paths ${ }_{\text {bool }}$ returns true on $\mathscr{D}$, so there is a sentence $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$ as above, which is true on $\mathscr{D}$; and implies $K$-Paths ${ }_{\text {bool }}$.

Let $W$ be the (constant) width of $\phi_{m}$; assume that $m \leqslant \gamma n^{K}$, where $\gamma$ is a constant to be chosen. We will show that Player II wins the weak version of the $(p, m, W)$-linear Ehrenfeucht-Fraïssé game with $\neq$ on the above structure $\mathscr{D}$, assuming that Player I plays a sequence of moves corresponding-as in the Proof of Theorem 17-to the formula $\phi_{m}$. Note the following:
(a) Since the $\phi_{k}$ 's do not have existential quantifiers, Player I plays each pebble exactly once. Also, when a pebble is played, it is included in the window of that round.
(b) For $k=1, \ldots, m$, every variable of $\beta_{k}$ which occurs in $\phi_{k-1}$, occurs in $\beta_{k-1}$. It follows that: any pebble in the window of the $k$ th round, which was played in a previous round (i.e., corresponding to a variable occurring in $\phi_{k-1}$ ), is also in the window of the $(k-1)$ st round.

We will use the above properties to show that Player II can win by picking a structure $\mathscr{D}^{\prime}$ such that: the query $K$ - Paths $_{\text {bool }}$ returns false on $\mathscr{D}^{\prime}$. Since Player II wins, the sentence $\exists v_{1} \cdots \exists v_{p} . \phi_{m}$ is true in $\mathscr{D}^{\prime}$.

Let the nodes of the $i$ th path of $\mathscr{D}$ be $e_{j}^{i}$, where $i=1, \ldots, K, j=1, \ldots, \frac{n}{K}$ (we have $e_{1}^{i}=a, e_{\bar{n}}^{i}=b$ ). A cut of $\mathscr{D}$ is a set of $K$ nodes containing exactly one node of each path of $\mathscr{D}$ (it cannot contain the endpoints $a$ and $b$ ).

We count the cuts that are entirely contained in a window designated by Player I: since each window consists of at most $W$ nodes, the number of cuts contained in one window is at most $W^{K}$; so the total number of cuts contained in any of the $m$ windows is at most $m W^{K}$, i.e., at most $\gamma n^{K} W^{K}$. The number of cuts of $\mathscr{D}$ is $\left(\frac{n}{K}\right)^{K}$; thus, if $\gamma<(K W)^{-K}$ there is some cut of $\mathscr{D}$ which is not entirely contained in any window designated by Player I.

Let $e_{j_{1}}^{1}, \ldots, e_{j_{K}}^{K}$ be such a cut. Player II picks a structure $\mathscr{D}^{\prime}$ constructed from $\mathscr{D}$ by removing the nodes of the above cut; inserting $K-1$ new nodes $\zeta^{1}, \ldots, \zeta^{K-1}$; and inserting the $\operatorname{arcs}\left(e_{j_{i}-1}^{i}, \zeta^{i^{\prime}}\right),\left(\zeta^{i^{\prime}}, e_{j_{i}+1}^{i}\right)$, for $i=1, \ldots, K, i^{\prime}=1, \ldots, K-1$. Clearly $\mathscr{D}^{\prime}$ does not have $K$ node-disjoint paths from $a$ to $b$-so the query $K$ - Paths ${ }_{\text {bool }}$ returns false on $\mathscr{D}^{\prime}$.

The winning strategy of Player II is as follows. If Player I plays on a node of $\mathscr{D}$ outside the cut, Player II responds on the same node of $\mathscr{D}^{\prime}$. If Player I plays on some node $e_{j_{I}}^{I}$ in the cut, Player II will respond on a node $\zeta^{I^{\prime}}$ which he chooses as follows:
(i) The node $e_{j_{I}}^{I}$ is already pebbled, by some pebble that is included in the window. Player II responds on the node $\zeta^{I^{\prime}}$ pebbled by his corresponding pebble.
(ii) No pebble included in the window has been placed yet on the node $e_{j_{I}}^{I}$. The window of the round contains $e_{j_{I}}^{I}$-by the above property (a) - and does not exhaust the cut. Thus, just before $e_{j_{I}}^{I}$ is played, the window intersects the cut at a subset $C$ of cardinality at most $K-2$. Consider the set of pebbles of Player I lying on $C$ at that point; the corresponding pebbles of Player II are lying on the $\zeta^{i^{\prime}}$ 's; in particular on a subset $C^{\prime}$ of the $\zeta^{i^{\prime}}$ 's of cardinality at most $K-2$. Player II chooses arbitrarily a node $\zeta^{I^{\prime}}$ not in $C^{\prime}$, and responds on $\zeta^{I^{\prime}}$.

The strategy of Player II maintains a one-to-one homomorphism of each window designated by Player I. To see this we argue by induction on the number of rounds. Suppose at round $k$ some of the pebbles played in previous rounds are designated in the window of round $k$. By the above property (b), the pebbles were designated in the window of round $k-1$; so by the inductive hypothesis the one-to-one homomorphism has been maintained on them. The choice of Player II maintains the one-to-one homomorphism between each new pebble in the window and the pebbles already played. Thus the one-to-one homomorphism is maintained on the entire window.

The above results can be adapted straightforwardly to Extended-Datalog.

### 4.2. Lower Bounds for Derivation Tree Size

Recall that upper bounds on the derivation tree size imply upper bounds on parallel complexity. In particular, a Datalog program with polynomial derivation tree size can be evaluated in $\mathscr{N} C$ [UvG88]. Since P-complete queries are not expected to be in $\mathscr{N C}$, one should also expect that they do not have polynomial derivation tree size.

We consider the prototypical P-complete query Access. It is expressed by the following Datalog program (on a database ( $D$, reach, source)).

$$
\begin{align*}
\operatorname{ACCESS}(x) \leftarrow & \operatorname{SOU} \operatorname{RCE}(x) \\
\operatorname{ACCESS}(x) \leftarrow & R E A C H(x, y, z),  \tag{6}\\
& A C C E S S(y), \operatorname{ACCESS}(z)
\end{align*}
$$

We will show the following lower bound:
Theorem 25. Every Datalog program that expresses Access has derivation tree size $2^{\Omega(\sqrt{n})}$.

It is well-known that Datalog does not express all queries in $\mathscr{N C}$ (a simple example is parity). Thus the above result does not separate $\mathscr{P}$ from $\mathscr{N} C$.

We first describe a technical tool. The reader familiar with the literature on conjunctive queries and Datalog will notice an analogy with canonical instances and expansions.

Definition 26. Let $\phi$ be a linear formula. As in Definition 11, $\phi \equiv \phi_{m}$, where

$$
\begin{aligned}
& \phi_{1} \equiv \beta_{1} \\
& \phi_{k} \equiv \beta_{k} \wedge \exists \bar{x}_{k} \cdot \phi_{k-1} \quad \text { for } \quad k=2, \ldots, m .
\end{aligned}
$$

We assume that $\beta_{k}, k=1, \ldots, m$, is a quantifier-free formula without disjunctions and equalities (and without negation).

The hypergraph of $\phi, \mathscr{H}_{\phi}$, is a structure over the same signature- $\left(R_{1}, \ldots, R_{N}\right)-$ as $\phi$.

The domain of $\mathscr{H}_{\phi}$ is $\bigcup_{k=1, \ldots, m} V_{k}$, where each $V_{k}$ is called a row of $\mathscr{H}_{\phi}$.
For each variable $x$ which occurs free in $\phi_{k}$, the row $V_{k}$ contains a corresponding element $e_{x}^{k}$; moreover, if $x$ also occurs free in $\phi_{k-1}$ we identify $e_{x}^{k}$ with $e_{x}^{k-1}$.
For each atomic formula $R\left(x_{1}, \ldots, x_{W}\right)$ in $\beta_{k}$, the structure $\mathscr{H}_{\phi}$ contains a tuple $r\left(e_{x_{1}}^{k}, \ldots, e_{x_{w}}^{k}\right)$.

Observe that the size of each row of $\mathscr{H}_{\phi}$ is bounded by the number of variables of $\phi$.

Remark 27. If a variable $x$ occurs free in $\phi_{k}$, let $k_{1}$ be the smallest index, after $k$, such that: $x$ is bound in the quantifier prefix $\exists \bar{x}_{k_{1}}$. Let $k_{2}$ be the largest index, before $k$, such that: $x$ does not occur free in $\exists \bar{x}_{k_{2}} \cdot \phi_{k_{2}-1}$.

The variable $x$ occurs free in every $\phi_{i}$ for $i=k_{2}, \ldots, k_{1}-1$; moreover, these are all the free occurences of $x$ with the same meaning as the one in $\phi_{k}$. We have

$$
e_{x}^{k_{2}}=\cdots=e_{x}^{k}=\cdots=e_{x}^{k_{1}-1}
$$

Also, if $i<k_{2}$ or $i \geqslant k_{1}$ we have $e_{x}^{i} \neq e_{x}^{k}$.

Definition 28. Let $\mathscr{D}=\left(D, r_{1}, \ldots, r_{N}\right)$ be a structure. A path of $\mathscr{D}$ is a sequence of elements of $D$ such that: if $e, e^{\prime}$ are consecutive in the sequence, they both occur in some tuple of some relation $r_{i}$.

Alternatively, this could have been defined as a path in the Gaifman graph [Gai82] of $\mathscr{D}$.

Lemma 29. Let $\mathscr{H}_{\phi}$ be the hypergraph of a linear formula $\phi$, as in Definition 26. Suppose $\mathscr{H}_{\phi}$ has a path connecting the elements $a$, $b$, where $a \in V_{k_{1}}, b \in V_{k_{2}}$.

Then, for each $k$ between $k_{1}$ and $k_{2}$, the path intersects the row $V_{k}$.
Proof. Observe the following:
If an element $e$ of $\mathscr{H}_{\phi}$ belongs to $V_{l_{1}}$ and $V_{l_{2}}$ then it also belongs to $V_{l}$, for each $l$ between $l_{1}$ and $l_{2}$ (see Remark 27).

If both elements $e, e^{\prime}$ occur in a tuple of $\mathscr{H}_{\phi}$, then they both belong to some $V_{k}$ (by Definition 26).

We can insert in the path appropriate repetitions of elements, so that in the resulting path: if $e, e^{\prime}$ are consecutive, then for some $k$

$$
\begin{array}{ll}
e, e^{\prime} \in V_{k}, \quad \text { or } \\
e \in V_{k-1}, e^{\prime} \in V_{k}, & \text { or } \\
e^{\prime} \in V_{k-1}, e \in V_{k} .
\end{array}
$$

The lemma follows.
Lemma 30. Let $\mathscr{H}_{\phi}$ be the hypergraph of a linear formula $\phi$, as in Definition 26. For $j=1, \ldots, n$, let $E_{j}$ be a path of $\mathscr{H}_{\phi}$. Let $S_{j}=\left\{k \mid\right.$ the row $V_{k}$ of $\mathscr{H}_{\phi}$ intersects $\left.E_{j}\right\}$.

Either (i) there is some integer $k_{0}$ which belongs to every $S_{j}$ or (ii) there are integers $j_{1}, j_{2}$ such that $S_{j_{1}}, S_{j_{2}}$ are disjoint.

Proof. Let $U_{j}, L_{j}$ be the largest and the smallest element of $S_{j}$, respectively. By Lemma 29, the set $S_{j}$ consists of the integers between $L_{j}$ and $U_{j}$.

We show that, for any $l$, the conclusion of the Lemma holds for the sets $\left\{S_{1}, \ldots, S_{l}\right\}$. We argue by induction on $l$; the base case is obvious.

Assume the conclusion of the Lemma for the sets $\left\{S_{1}, \ldots, S_{l}\right\}$. If (ii) holds, then clearly (ii) holds for the sets $\left\{S_{1}, \ldots, S_{l}, S_{l+1}\right\}$.

If (i) holds for the sets $\left\{S_{1}, \ldots, S_{l}\right\}$, let $S=\bigcap_{j=1, \ldots, l} S_{j}$. If $S_{l+1}$ intersects $S$, clearly (i) holds for the sets $\left\{S_{1}, \ldots, S_{l}, S_{l+1}\right\}$.

If $S_{l+1}$ does not intersect $S$, let $U, L$ be the largest and the smallest element of $S$, respectively. We have $U_{l+1}<L$ or $L_{l+1}>U$. Now there exist integers $j_{1}$ and $j_{2}-$ between 1 and $l-$ such that $U_{j_{1}}=U, L_{j_{2}}=L$. Thus, either $U_{l+1}<L_{j_{2}}$, which means that $S_{l+1}, S_{j_{2}}$ are disjoint; or $L_{l+1}>U_{j_{1}}$, which means that $S_{l+1}, S_{j}$, are disjoint.

Let $\phi\left(x_{1}, \ldots, x_{w}\right)$ be a formula with free variables $x_{1}, \ldots, x_{w}$. We denote by $\phi\left(a_{1}, \ldots, a_{w}\right)$ the result (truth value) of assigning the value $a_{i}$ to the variable $x_{i}$ of $\phi$.

Lemma 31. Let $\phi$ be a linear formula with free variables $x_{1}, \ldots, x_{w}$; let $e_{x_{i}}$ be the element of $\mathscr{H}_{\phi}$ corresponding to the free occurrence of $x_{i}$ in $\phi$.
(i) Let $\mathscr{D}$ be a database and let $a_{1}, \ldots, a_{w}$ be vales of $\mathscr{D}$. Then $\phi\left(a_{1}, \ldots, a_{w}\right)$ holds in $\mathscr{D}$ iff: there is a homomorphism $h$ from $\mathscr{H}_{\phi}$ to $\mathscr{D}$ such that $h\left(e_{x_{i}}\right)=a_{i}$.
(ii) $\phi\left(e_{x_{1}}, \ldots, e_{x_{w}}\right)$ holds in $\mathscr{H}_{\phi}$.

## Proof. (i) Straightforward.

(ii) Apply (i); consider the identity homomorphism from $\mathscr{H}_{\phi}$ to $\mathscr{H}_{\phi}$.

Recall that the linear formulas constructed in the Proof of Theorem 15 do not contain disjunctions or equalities. Theorem 25 follows from Theorem 15 and the following result:

Theorem 32. Every sequence of linear formulas without disjunctions and equalities that expresses Access has $\Omega(\sqrt{n})$ variables.
Proof. We consider the following database $\mathscr{D}_{n}=\left(D_{n}\right.$, reach $_{n}$, source $\left._{n}\right)$, for arbitrary $n$ :

$$
\begin{aligned}
D_{n} & =\{(i, j) \mid i=0, \ldots, n, j=0, \ldots, n\}-\{(0,0)\} . \\
\text { source }_{n} & =\{(i, 0) \mid i=1, \ldots, n\} \cup\{(0, j) \mid j=1, \ldots, n\} . \\
\text { reach }_{n} & =\{\langle(i, j),(i-1, j),(i, j-1)\rangle \mid i=1, \ldots, n, j=1, \ldots, n\} .
\end{aligned}
$$

If the Datalog program 6 is evaluated on $\mathscr{D}_{n}$, the resulting relation $\operatorname{ACCESS}$ is $\{(i, j) \mid i=1, \ldots, n, j=1, \ldots, n\}$.

Let $\phi(x)$ be a formula of the sequence (with a free variable $x$ ) with $M$ variables, such that: (i) $\phi((n, n))$ holds in $\mathscr{D}_{n}$; and (ii) for any database $\mathscr{D}$, if $\phi(a)$ holds in $\mathscr{D}$ then $a \in \operatorname{Access}(\mathscr{D})$. We will show that $M \geqslant n$; the Theorem follows, since $\mathscr{D}_{n}$ has $(n+1)^{2}-1$ elements.

We consider the structure $\mathscr{H}_{\phi}=(H$, reach, source $)$; we will show that there is a row of $\mathscr{H}_{\phi}$ with at least $n$ elements.

Let $e$ be the element of $\mathscr{H}_{\phi}$ corresponding to the free occurrence of $x$ in $\phi$. By Lemma 31(i), there is a homomorphism $h$ from $\mathscr{H}_{\phi}$ to $\mathscr{D}_{n}$ such that $h(e)=(n, n)$.

By Lemma 31(ii), $\phi(e)$ holds in $\mathscr{H}_{\phi}$. Thus, $e \in \operatorname{Access}\left(\mathscr{H}_{\phi}\right)$.
Since $e \in \operatorname{Access}\left(\mathscr{H}_{\phi}\right)$, we have either $e \in$ source, or $\left\langle e, e_{1}, e_{2}\right\rangle \in$ reach-where $e_{1}, e_{2} \in \operatorname{Access}\left(\mathscr{H}_{\phi}\right)$; since $h(e)=(n, n)$ and $(n, n) \notin$ source $_{n}$, we cannot have $e \in$ source. Now since $\left\langle e, e_{1}, e_{2}\right\rangle \in$ reach it must be $\left\langle h(e), h\left(e_{1}\right), h\left(e_{2}\right)\right\rangle \in$ reach $_{n}$; therefore $h\left(e_{1}\right)=(n-1, n), h\left(e_{2}\right)=(n, n-1)$.

We can keep repeating this argument-with $e_{1}$ and $e_{2}$ in the place of $e$, and so on. It follows that $\mathscr{H}_{\phi}$ has a substructure isomorphic to $\mathscr{D}_{n}$. To avoid additional notation, we assume (with no loss of generality) that $\mathscr{D}_{n}$ is a substructure of $\mathscr{H}_{\phi}$.

Consider the set $E_{j}=\{(i, j) \mid i=0, \ldots, n\}$ of elements of $\mathscr{D}_{n}$, where $j=1, \ldots, n$. The elements of $E_{j}$ form a path of $\mathscr{H}_{\phi}$.

Let $S_{j}=\left\{k \mid\right.$ the row $V_{k}$ of $\mathscr{H}_{\phi}$ intersects $\left.E_{j}\right\}$; by Lemma 30, either (i) there is some integer $k_{0}$ which belongs to every $S_{j}$, or (ii) there are integers $j_{1}, j_{2}$ such that $S_{j_{1}}, S_{j_{2}}$ are disjoint.
If (i) holds, the row $V_{k_{0}}$ intersects every path $E_{j}$; thus $V_{k_{0}}$ has at least $n$ elements, since the $E_{j}$ 's are mutually disjoint.

If (ii) holds, there are integers $j_{1}, j_{2}, k_{0}$ such that no element of $S_{j_{1}}$ is larger than $k_{0}$; and no element of $S_{j_{2}}$ is smaller than $k_{0}$. Also, $j_{1}<j_{2}$.

Consider the set $F_{i}=\left\{(i, j) \mid j=j_{1}, \ldots, j_{2}\right\}$ of elements of $\mathscr{D}_{n}$, where $i=1, \ldots, n$. The elements of $F_{i}$ form a path of $\mathscr{H}_{\phi}$. Now let $\left(i, j_{1}\right) \in V_{k_{1}},\left(i, j_{2}\right) \in V_{k_{2}}$. Since $\left(i, j_{1}\right) \in E_{j_{1}}$, we have $k_{1} \in S_{j_{1}}$; similarly, $k_{2} \in S_{j_{2}}$. By the choice of $j_{1}, j_{2}, k_{0}$, we have $k_{1} \leqslant k_{0} \leqslant k_{2}$. It follows by Lemma 29 that the path $F_{i}$ intersects the row $V_{k_{0}}$. Since the $F_{i}$ 's are mutually disjoint, $V_{k_{0}}$ has at least $n$ elements.

Remark 33. There are several Datalog queries that have been shown to be P-complete [UvG88, AP93]. The reductions are from simple variants of Access. Moreover, they are first-order reductions in the sense of [Imm87], and in particular the formulas they use are quantifier-free. It follows that lower bounds similar to Theorem 25 can be shown for the P-complete Datalog queries of [UvG88, AP93]. We leave the details to the interested reader.

The Datalog program 6 has derivation trees of size $2^{O(n)}$; by Theorem 15, Access can by expressed by a sequence of linear formulas with $O(n)$ variables. These upper bounds are not matched by the lower bounds in Theorem 25 and Theorem 32.

We will show tighter lower bounds for databases with few paths.
Definition 34. A database $\mathscr{D}$ is tree-like if the total number of paths of $\mathscr{D}$ is at most polynomial (in the size of $\mathscr{D}$ ).

Theorem 35. The Datalog program 6 has polynomial-size derivation trees on treelike databases.

Proof. Let $\mathscr{D}$ be a database of size $n$, and let $a \in \operatorname{Access}(\mathscr{D})$. Since the IDB-predicate $A C C E S S$ is unary, the recursively defined relation $A C C E S S$ has at most $n$ elements. Thus, there exists a derivation dag $\delta$ with root $u$ labeled $\operatorname{ACCESS}(a)$, and with at most $n$ nodes.

Let $f$ be a node of $\delta$; let $F$ be a path of $\delta$, from $f$ to the root $u$. It is easy to see that: the elements in the labels of the path $F$ form a path of $\mathscr{D}$ (because of the form of the rules of program 6). Since $\mathscr{D}$ is tree-like, it has at most a polynomial number of distinct paths. Also, the predicate in each label of the path $F$ is $A C C E S S$-with the possible exception of the label of $f$, where the predicate can also be $S O U R C E$.

Thus, there is a polynomial-size set $X_{f}$ of sequences of labels such that: each distinct sequence of labels of a path from $f$ to $u$ appears once in $X_{f}$; and these are the only sequences in $X_{f}$.

We now construct a polynomial-size derivation tree $\tau$ with root labeled $\operatorname{ACCESS}(a)$.

The nodes of $\tau$ are the pairs $\langle f, \sigma\rangle$, where $f$ is a node of $\delta$ and $\sigma \in X_{f}$. The label of $\langle f, \sigma\rangle$ in $\tau$ is the same as the label of $f$ in $\delta$.

Let $\left(f_{1}, f\right),\left(f_{2}, f\right)$ be the arcs of $\delta$ coming into the node $f$; let $g_{1}, g_{2}$ be the labels of $f_{1}, f_{2}$ respectively. For each node $\langle f, \sigma\rangle$ of $\tau$, we put arcs

$$
\left(\left\langle f_{1}, g_{1} \sigma\right\rangle,\langle f, \sigma\rangle\right) \quad \text { and } \quad\left(\left\langle f_{2}, g_{2} \sigma\right\rangle,\langle f, \sigma\rangle\right)
$$

in $\tau$; where $g_{i} \sigma$ is the sequence of labels obtained by inserting $g_{i}$ in the front of the sequence $\sigma$.

It is easy to check that $\tau$ is a derivation tree with the desired properties.
It is not clear how to characterize the class of tree-like databases in terms of structural properties. For example, it is easy to see that it is incomparable with the class of bounded tree-width (or bounded path-width) graphs. Similarly, the above result does not hold for graphs of bounded path-width (consider a graph obtained by collating a large number of small cliques, so that each clique shares one node with the next clique in the sequence).

By the above result and Theorem 15, Access can be expressed on tree-like databases by a sequence of linear formulas with $O(\log n)$ variables.

We show the following lower bounds:
Theorem 36. (i) Every sequence of linear formulas without disjunctions and equalities that expresses Access on tree-like databases has $\Omega(\log n)$ variables.
(ii) Every Datalog program that expresses Access on tree-like databases has derivation tree size $n^{\Omega(1)}$.

Proof. Recall that the linear formulas constructed in the Proof of Theorem 15 do not contain disjunctions or equalities. Thus, (ii) follows from (i) and Theorem 15.

Given a database $\mathscr{D}=(D$, reach, source $)$, let $\mathscr{T}$ be an undirected graph constructed as follows: the set of nodes of $\mathscr{T}$ is $D$; for each tuple $\left\langle e, e_{1}, e_{2}\right\rangle$ in reach, $\mathscr{T}$ contains edges $\left\{e, e_{1}\right\}$ and $\left\{e, e_{2}\right\}$. In the rest of this Proof, the statement " $\mathscr{D}$ is a tree" will mean "the graph $\mathscr{T}$ is a tree."

To prove (i): we consider a tree-like database $\mathscr{D}_{n}=\left(D_{n}\right.$, reach $_{n}$, source $\left._{n}\right)$, for arbitrary $n$. The database $\mathscr{D}_{n}$ is a rooted tree; its root is the element $a^{n}$.
$\mathscr{D}_{0}$ consists of a single element $a^{0} ;$ source $_{0}=\left\{a^{0}\right\}$, reach $_{0}=\varnothing$.
$\mathscr{D}_{i+1}$ contains three disjoint copies of $\mathscr{D}_{i}$. For $l=1,2,3$, the $l$ th copy is a tree $\mathscr{D}_{i, l}$ (isomorphic to $\mathscr{D}_{i}$ ) with root $a_{l}^{i}$. In addition, $D_{i+1}$ contains two nodes $a^{i+1}, b$; and reach ${ }_{i+1}$ contains the tuples $\left\langle a^{i+1}, b, a_{1}^{i}\right\rangle,\left\langle b, a_{2}^{i}, a_{3}^{i}\right\rangle$.

Clearly, $a^{n} \in \operatorname{Access}\left(\mathscr{D}_{n}\right)$.
Let $\phi(x)$ be a formula of the sequence (with a free variable $x$ ) with $M$ variables, such that: (i) $\phi\left(a^{n}\right)$ holds in $\mathscr{D}_{n}$; and (ii) for any database $\mathscr{D}$, if $\phi(a)$ holds in $\mathscr{D}$ then $a \in \operatorname{Access}(\mathscr{D})$. We will show that $M \geqslant n$; the result follows, since $\mathscr{D}_{n}$ has $2 \times 3^{n}-1$ elements.

We consider the structure $\mathscr{H}_{\phi}=(H$, reach, source $)$; we will show that there is a row of $\mathscr{H}_{\phi}$ with at least $n$ elements.

Arguing as in the Proof of Theorem 32, we show that $\mathscr{H}_{\phi}$ has a substructure isomorphic to $\mathscr{D}_{n}$. To avoid additional notation, we assume (with no loss of generality) that $\mathscr{D}_{n}$ is a substructure of $\mathscr{H}_{\phi}$.

Claim. Suppose $\mathscr{D}_{i}$ is a substructure of $\mathscr{H}_{\phi}$, where $i \geqslant 0$. Then there is a row of $\mathscr{H}_{\phi}$ which contains at least i elements of $\mathscr{D}_{i}$.

Proof of Claim. We argue by induction on $i$; the base case is obvious.
$\mathscr{D}_{i+1}$ is a rooted tree with root $a^{i+1}$. It contains three disjoint copies of $\mathscr{D}_{i}$; the $l$ th copy is a subtree $\mathscr{D}_{i, l}$ of $\mathscr{D}_{i+1}$, with root $a_{l}^{i}$.

Let $l=1,2,3$. Apply the inductive hypothesis to the subtree $\mathscr{D}_{i, l}$ rooted at $a_{l}^{i}$, to obtain a row $V_{k_{l}}$ of $\mathscr{H}_{\phi}$ containing at least $i$ elements of $\mathscr{D}_{i, l}$.

If any two of the indices $k_{1}, k_{2}, k_{3}$ coincide, there is a single row of $\mathscr{H}_{\phi}$ which contains at least $2 i$ elements of $\mathscr{D}_{i+1}$ (since the subtrees $\mathscr{D}_{i, l}$ are disjoint).

If $k_{1}, k_{2}, k_{3}$ are all different, assume (with no loss of generality) that $k_{1}<k_{2}<k_{3}$.
Let $V_{k_{0}}$ be a row containing the element $a^{i+1}$. If $k_{0} \leqslant k_{2}$, consider a path of $\mathscr{D}_{i+1}$ from the root $a^{i+1}$ to an element of $\mathscr{D}_{i, 3}$ contained in $V_{k_{3}}$. Note that this path contains no elements of $\mathscr{D}_{i, 2}$. By Lemma 29, the path intersects the row $V_{k_{2}}$. So $V_{k_{2}}$ contains at east one element of $\mathscr{D}_{i+1}$ which is not in $\mathscr{D}_{i, 2}$, i.e., $V_{k_{2}}$ contains at least $i+1$ elements of $\mathscr{D}_{i+1}$.

If $k_{0} \geqslant k_{2}$, we argue similarly, considering a path of $\mathscr{D}_{i+1}$ from the root $a^{i+1}$ to an element of $\mathscr{D}_{i, 1}$ contained in $V_{k_{1}}$.

It follows from the Claim that there is a row of $\mathscr{H}_{\phi}$ with at least $n$ elements.
The results in this Subsection can be extended straightforwardly to $\operatorname{Datalog}(\neq)$; and furthermore to $\operatorname{Datalog}(\neq, \neg)$, which in addition allows negation on EDBpredicates.

## 5. LOWER BOUNDS FOR CHAIN QUERIES

In this Section we give lower bounds for the size of the recursively defined relations and the derivation tree size, for certain Datalog queries which ask about the existence of paths in directed graphs.

Definition 37. Let $\Lambda$ be a language over the alphabet $1, \ldots, s$. The chain query $Q_{A}$ on a database ( $D, \operatorname{arc}_{1}, \ldots, \operatorname{arc} c_{s}$ ) returns the set of tuples $\langle a, b\rangle$ such that: the labeled directed graph represented by the binary relations $\operatorname{arc}_{1}, \ldots, \operatorname{arc}_{s}$ has a path from $a$ to $b$, with sequence of labels corresponding to a word in $\Lambda$ (the label $\operatorname{arc}_{t}$ corresponds to the letter $t$ ).

Definition 38. A database ( $D, \operatorname{arc}_{1}, \ldots, \operatorname{arc} c_{s}$ ) is acyclic if the labeled directed graph represented by the binary relations $\operatorname{arc}_{1}, \ldots, \operatorname{arc}_{s}$ is acyclic.

The following is implicit in [KV95].
Proposition 39. (i) Every chain query can be expressed on acyclic databases by a sequence of linear formulas with depth $O(n)$.
(ii) Every chain query can be expressed by a sequence of linear formulas with a constant number of variables.

By the above observations, we cannot use Theorem 12 to show nontrivial lower bounds for the derivation dag size of chain queries, ${ }^{9}$ on acyclic databases; nor can we use Theorem 15 to show nontrivial lower bounds for the derivation tree size of chain queries.

[^5]We will prove such lower bounds, by exploiting the uniformity of the sequences of linear formulas (in Theorems 12 and 15) which express a Datalog query.

We denote the empty word by $\epsilon$; the concatenation of the words $\omega_{1}, \omega_{2}$ by $\omega_{1} \omega_{2}$; the $n$-fold concatenation $\underbrace{\omega \cdots \omega}_{n \text { times }}$ by $\omega^{n}\left(l^{n}\right.$ is the word of $n \imath$ 's); and the length of the word $\omega$ by $|\omega|$.

We will show that, for arbitrary $K$, there exists a chain query definable in Datalog such that the size of the recursively defined relations is $\Omega\left(n^{K}\right)$, even on acyclic databases.

Definition 40. The four same paths query 4-SamePaths on a database ( $D, \operatorname{arc}_{1}$ ) returns the set of tuples $\left\langle a_{1}, a_{1}^{\prime}, a_{2}, a_{2}^{\prime}, a_{3}, a_{3}^{\prime}, a_{4}, a_{4}^{\prime}\right\rangle$ such that: there are four paths of equal length in the relation $\operatorname{arc} c_{1}$ connecting, respectively, $a_{1}$ to $a_{1}^{\prime}, a_{2}$ to $a_{2}^{\prime}, a_{3}$ to $a_{3}^{\prime}$, and $a_{4}$ to $a_{4}^{\prime}$.

The query 4-SamePaths is expressed by the following Datalog program:

$$
\begin{align*}
& \text { 4-SAMEPATHS }\left(x, x^{\prime}, y, y^{\prime}, z, z^{\prime}, w, w^{\prime}\right) \\
& \leftarrow x=x^{\prime}, y=y^{\prime}, z=z^{\prime}, w=w^{\prime} \\
& \text { 4-SAMEPATHS}\left(x, x^{\prime}, y, y^{\prime}, z, z^{\prime}, w, w^{\prime}\right)  \tag{7}\\
& \leftarrow A R C_{1}(x, t), \operatorname{ARC} C_{1}(y, u), A R C_{1}(z, v), A R C_{1}(w, g), \\
& \text { 4-SAMEPATHS}\left(t, x^{\prime}, u, y^{\prime}, v, z^{\prime}, g, w^{\prime}\right)
\end{align*}
$$

Definition 41. (i) Let $K$ be a fixed integer. The $K$ equal paths query $K$-EqualPaths on a database ( $D, \operatorname{arc}_{1}, \operatorname{arc}_{2}$ ) returns the set of tuples $\left\langle a_{1}, a_{1}^{\prime}, \ldots, a_{K}, a_{K}^{\prime}\right\rangle$ such that: there are $K$ paths of equal length connecting, respectively, $a_{1}$ to $a_{1}^{\prime}, \ldots, a_{K}$ to $a_{K}^{\prime}$; the arcs of the $i$ th path are labeled by $\operatorname{arc} c_{1}$ for $i$ odd, and by $\operatorname{arc}_{2}$ for $i$ even.
(ii) Let $K$ be a fixed even integer. The $K$ equalities query $K$-Equals on a database ( $D, \operatorname{arc}_{1}, \operatorname{arc_{2}}$ ) is the chain query $Q_{\Lambda_{K}}$ obtained from the following language $\Lambda_{K}$ :

$$
\underbrace{1^{n} 2^{n} \cdots 1^{n} 2^{n}}_{\frac{K}{2} \text { times }} \in \Lambda_{K} .
$$

The query $K$-Equal Paths can be expressed in Datalog; the defining program is similar to program 7.

The query $K$-Equals is expressed by the following Datalog program.

$$
\begin{equation*}
K-E Q U A L S(x, y) \leftarrow K-E Q U A L P A T H S\left(x, z_{1}, z_{1}, z_{2}, \ldots, z_{K-1}, y\right) \tag{8}
\end{equation*}
$$

Note that, since the Datalog program 8 is linear, a minimal derivation tree for the program is also a derivation dag.

Proposition 42. (i) The Datalog program 8 has derivation trees of size $O\left(n^{K}\right)$.
(ii) The Datalog program 8 has derivation trees of size $O(n)$ on acyclic databases.

Proof. (i) It suffices to show that the Datalog program for $K$-EqualPaths has derivation trees of size $O\left(n^{K}\right)$.

Consider the program 7; the size of a derivation tree is bounded by the maximum possible number of tuples $\langle t, u, v, g\rangle$, i.e., it is $O\left(n^{4}\right)$. The general case is similar.
(ii) Let $\mathscr{D}$ be an acyclic database with $n$ elements, and $a, b$ elements of $\mathscr{D}$ such that $a, b \in K-E q u a l s(\mathscr{D})$. There is a path (of $\mathscr{D}$ ) from $a$ to $b$, labeled by a word $\omega \in \Lambda_{K}$. It is easy to see that there is a derivation tree with root labeled $K-E Q U A L S(a, b)$; and with size $O(|\omega|)$. Since $\mathscr{D}$ is acyclic, $|\omega| \leqslant n$.

By the above observations, the size of the recursively defined relations for program 8 is $O\left(n^{K}\right)$.

We will show the following lower bound:
Theorem 43. For every Datalog program that expresses $K$-Equals on acyclic databases, the size of the recursively defined relations is $\Omega\left(n^{K}\right)$.

Proof. From a given Datalog program $\Pi$ that expresses $K$-Equals on acyclic databases we construct an acyclic database $\mathscr{D}_{1}=\left(D_{1}, \operatorname{arc} c_{1,1}, \operatorname{arc} c_{2,1}\right)$. The database $\mathscr{D}_{1}$ is a simple path connecting the elements $a, b$. The construction uses an integer $\Gamma$ depending on the program $\Pi$.
$\mathscr{D}_{1}$ consists of $K$ (mutually disjoint, simple) subpaths of length $\Gamma$. For $i=1, \ldots, K$ the $i$ th subpath, $E_{1}^{i}$, connects the elements $\zeta_{i-1}, \zeta_{i}$ (where $\left.\zeta_{0} \equiv a, \zeta_{K} \equiv b\right)$. The arcs of the subpath $E_{1}^{i}$ are labeled by $\operatorname{arc}_{1}$ for $i$ odd, and by $\operatorname{arc}_{2}$ for $i$ even.

Clearly, $\langle a, b\rangle \in K-E q u a l s\left(\mathscr{D}_{1}\right)$.
We consider a sequence of linear formulas constructed from the program $\Pi$, as in the Proof of Theorem 12; recall that these formulas do not contain disjunctions or equalities. Let $\phi(x, y)$ be a formula of the sequence (with free variables $x, y$ ) such that: (i) $\phi(a, b)$ holds in $\mathscr{D}_{1}$; and (ii) for any database $\mathscr{D}$, if $\phi(a, b)$ holds in $\mathscr{D}$ then $\langle a, b\rangle \in K$-Equals( $\mathscr{D})$.

We consider the structure $\mathscr{H}_{\phi}=\left(H, \operatorname{arc}_{1}, \operatorname{arc}_{2}\right)$.
Arguing as in the Proof of Theorem 32, we show that $\mathscr{H}_{\phi}$ has at least one substructure isomorphic to $\mathscr{D}_{1}$. To avoid additional notation, we assume (with no loss of generality) that $\mathscr{D}_{1}$ is a substructure of $\mathscr{H}_{\phi}$.

Lemma. There is some substructure of $\mathscr{H}_{\phi}$ isomorphic to $\mathscr{D}_{1}$ such that: there is a row of $\mathscr{H}_{\phi}$ intersecting every path $E_{1}^{i}$ of $\mathscr{D}_{1}$.

Proof of Lemma. Assume it is false; we will derive a contradiction.
Consider an arbitrary copy of $\mathscr{D}_{1}$ in $\mathscr{H}_{\phi}$. The statement of the Lemma is violated. By Lemmas 29 and 30, there is a row $V_{k_{0}}$ which "separates" two of the paths in the statement-say, without loss of generality, the path $E_{1}^{1}$ is separated from the path $E_{1}^{2}$. More precisely, this means that if the row $V_{k}$ of $\mathscr{H}_{\phi}$ intersects the path $E_{1}^{1}$, we have $k<k_{0}$; and if $V_{k}$ intersects the path $E_{1}^{2}$, we have $k_{0} \leqslant k$.

By a straightforward application of the "pumping" technique of [AC89, ACY95] (see [ACY95], Lemma 4.8) to the part of $\mathscr{H}_{\phi}$ consisting of the rows $V_{k}$ with $k<k_{0}$, we can show the following. If $\Gamma$ is sufficiently large with respect to the
program, there is some formula $\phi^{\prime}$ of the sequence, such that $\mathscr{H}_{\phi^{\prime}}$ has the following property: $\mathscr{H}_{\phi^{\prime}}$ contains a path from $a$ to $b$, which is obtained by repeating-an arbitrary number of times - certain subpaths (see [ACY95], Definition 4.7) of the path from $a$ to $b$ contained in $\mathscr{H}_{\phi}$. Specifically, by the choice of $k_{0}$, the subpaths we repeat include some subpaths of the path $E_{1}^{1}$ from $a$ to $\zeta_{1}$; and include no subpath of the path $E_{1}^{2}$ from $\zeta_{1}$ to $\zeta_{2}$. It is not hard to see (since the repetition can be done an arbitrary number of times) that $\mathscr{H}_{\phi^{\prime}}$ can be chosen so that the labels of the path from $a$ to $b$ in $\mathscr{H}_{\phi^{\prime}}$ do not correspond to a word in $\Lambda_{K}$.

By repeating this argument for each copy of $\mathscr{D}_{1}$ in $\mathscr{H}_{\phi}$, we obtain a structure $\mathscr{H}_{\psi}$, where $\psi(x, y)$ is some formula of the sequence; and if $e_{x}, e_{y}$ are the elements corresponding to the free occurences of $x, y$ in $\psi$ : there is no path from $e_{x}$ to $e_{y}$ in $\mathscr{H}_{\psi}$ with labels corresponding to a word in $\Lambda_{K}$. However, $\psi\left(e_{x}, e_{y}\right)$ holds in $\mathscr{H}_{\psi}$ (Lemma 31), so $\left\langle e_{x}, e_{y}\right\rangle \in K-\operatorname{Equals}\left(\mathscr{H}_{\psi}\right)$; this is a contradiction.

Recall that the formula $\phi$ is constructed from a derivation dag $\delta$ of $\Pi$ (Proof of Theorem 12), and the arcs of $\mathscr{H}_{\phi}$ associated with a row (Definition 26) correspond to the body of some rule of $\Pi$. Suppose $\mathscr{D}_{1}$ is a substructure of $\mathscr{H}_{\phi}$ as in the Lemma. Then there is a row of $\mathscr{H}_{\phi}$ intersecting every path $E_{1}^{i}$ of $\mathscr{D}_{1}$. It is easy to see that there is a label of (some node of) $\delta$ containing an element $\mu_{1}^{i}$ of the path $E_{1}^{i}$ of $\mathscr{D}_{1}$-where $i=1, \ldots, K$. Thus, there is a tuple of the recursively defined relations containing every element $\mu_{1}^{i}$ of $\mathscr{D}_{1}$.

We now construct an acyclic database $\mathscr{D}_{n}=\left(D_{n}, \operatorname{arc}_{1, n}, \operatorname{arc}_{2, n}\right)$, for arbitrary $n$. The database $\mathscr{D}_{n}$ consists of a set of paths connecting the elements $a, b$; and contains a copy of $\mathscr{D}_{1}$. The construction uses the integer $\Gamma$ (depending on the program $\Pi$ ).
$\mathscr{D}_{n}$ consists of $K n$ (mutually disjoint, simple) paths of length $\Gamma$. For $i=1, \ldots, K$, the paths $E_{j}^{i}, j=1, \ldots, n$, connect the elements $\zeta_{i-1}, \zeta_{i}$ (where $\zeta_{0} \equiv a, \zeta_{K} \equiv b$ ). The arcs of the paths $E_{j}^{i}$ are labeled by $\operatorname{arc}_{1}$ for $i$ odd, and by $\operatorname{arc}_{2}$ for $i$ even.

Clearly, $\langle a, b\rangle \in K-\operatorname{Equals}\left(\mathscr{D}_{n}\right)$.
We will show that there are $\Omega\left(n^{K}\right)$ tuples of the recursively defined relations; Theorem 43 follows, since $\mathscr{D}_{n}$ has $K \Gamma n$ elements.

Consider the element $\mu_{1}^{i}$ of $\mathscr{D}_{1}$. For $j=1, \ldots, n$, let $\mu_{j}^{i}$ be: the element of $E_{j}^{i}$ lying at the same distance from $\zeta_{i-1}$ as $\mu_{1}^{i}$.

Fix a sequence $j_{1}, \ldots, j_{K}$, where $j_{i}=1, \ldots, n$. There is an automorphism of $\mathscr{D}_{n}$ which maps $\mu_{1}^{i}$ to $\mu_{j_{i}}^{i}$. It follows that there is a tuple of the recursively defined relations containing the elements $\mu_{j_{i}}^{i}$ of $\mathscr{D}_{n}$, where $i=1, \ldots, n$.

Since there are $n^{K}$ sequences $j_{1}, \ldots, j_{K}$ and the arity of the IDB-predicates of $\Pi$ is bounded, there are $\Omega\left(n^{K}\right)$ tuples of the recursively defined relations.

We will now show that there exists a chain query definable in Datalog such that the size of derivation trees is $n^{\Omega(1)}$, even on acyclic databases.

Definition 44. The chain version of path system accessibility $C$-Access on a database ( $D, \operatorname{arc_{1}}, \operatorname{arc_{2}}$ ) is the chain query $Q_{\Lambda_{C}}$ obtained from the following language $\Lambda_{C}$ :

$$
\begin{aligned}
& \epsilon \in \Lambda_{C} \\
& \text { If } \omega_{1}, \omega_{2} \in \Lambda_{C} \text { then } 1^{n} 2 \omega_{1} 21^{n} 21^{n} 2 \omega_{2} 21^{n} \in \Lambda_{C}
\end{aligned}
$$

The query $C$-Access is expressed by the following Datalog program.

$$
\begin{align*}
C-\operatorname{ACCESS}(x, y) \leftarrow & x=y \\
C-\operatorname{ACCESS}(x, y) \leftarrow & 4-E Q U A L P A T H S\left(x, x_{2}, z_{2}, z_{1}, w_{1}, w_{2}, y_{2}, y\right), \\
& A R C_{2}\left(z_{1}, w_{1}\right), \\
& A R C_{2}\left(x_{2}, x^{\prime}\right), A R C_{2}\left(z^{\prime}, z_{2}\right),  \tag{9}\\
& C-A C C E S S\left(x^{\prime}, z^{\prime}\right), \\
& A R C_{2}\left(w_{2}, w^{\prime}\right), A R C_{2}\left(y^{\prime}, y_{2}\right), \\
& C-A C C E S S\left(w^{\prime}, y^{\prime}\right)
\end{align*}
$$

Proposition 45. The Datalog program 9 has derivation trees of size $O(n)$ on acyclic databases.

Proof. Let $\mathscr{D}$ be an acyclic database with $n$ elements, and $a, b$ elements of $\mathscr{D}$ such that $a, b \in C$ - $\operatorname{Access}(\mathscr{D})$. There is a path (of $\mathscr{D}$ ) from $a$ to $b$, labeled by a word $\omega \in \Lambda_{C}$. It is easy to see that there is a derivation tree with root labeled $C-\operatorname{ACCESS}(a, b)$; and with size $O(|\omega|)$. Since $\mathscr{D}$ is acyclic, $|\omega| \leqslant n$.

We show the following lower bound:
Theorem 46. Every Datalog program that expresses C-Access on acyclic databases has derivation tree size $n^{\Omega(1)}$.

Proof. From a given Datalog program $\Pi$ that expresses $C$-Access on acyclic databases we construct an acyclic database $\mathscr{D}_{n}=\left(D_{n}, \operatorname{arc}_{1, n}, \operatorname{arc} c_{2, n}\right)$, for arbitrary $n$. The database $\mathscr{D}_{n}$ is a simple path connecting the elements $a^{n}, b^{n}$. The construction uses an integer $\Gamma$ depending on the program $\Pi ; \Gamma$ does not depend on $n$.
$\mathscr{D}_{0}$ consists of a single element $a^{0} ; \operatorname{arc}_{1,0}=\operatorname{arc}_{2,0}=\varnothing$. We set $b^{0} \equiv a^{0}$.
$\mathscr{D}_{i+1}$ consists of three disjoint copies of $\mathscr{D}_{i}$; for $l=1,2,3$, the $l$ th copy is a path connecting the elements $a_{l}^{i}, b_{l}^{i}$. In addition, $D_{i+1}$ contains the nodes $a^{i+1}, b^{i+1}$; the nodes $\chi, \chi^{\prime}, c_{1}, c_{1}^{\prime}, c_{2}, c_{2}^{\prime}$; and additional nodes to form the following (mutually disjoint) paths:

Four paths of length $\Gamma$, with arcs labeled $\operatorname{arc}_{1}$, connecting $a^{i+1}$ to $c_{1} ; c_{1}^{\prime}$ to $\chi ; \chi^{\prime}$ to $c_{2}$; and $c_{2}^{\prime}$ to $b^{i+1}$ respectively.

A path of length 2 , with arcs labeled $\operatorname{ar} c_{2}$, connecting $c_{1}$ to $a_{1}^{i}$.
A path of length 3 , with arcs labeled $\operatorname{arc} c_{2}$, connecting $b_{1}^{i}$ to $a_{2}^{i}$.
A path of length 2 , with arcs labeled $\operatorname{arc}_{2}$, connecting $b_{2}^{i}$ to $c_{1}^{\prime}$.
The following arcs labeled $\operatorname{arc}_{2}:\left\langle\chi, \chi^{\prime}\right\rangle,\left\langle c_{2}, a_{3}^{i}\right\rangle,\left\langle b_{3}^{i}, c_{2}^{\prime}\right\rangle$.

From the definition of $\Lambda_{C}$ : if $\omega_{1}, \omega_{2}, \omega_{3} \in \Lambda_{C}$, we have $2 \omega_{1} 222 \omega_{2} 2 \in \Lambda_{C}$; and therefore

$$
1^{\Gamma} 22 \omega_{1} 222 \omega_{2} 221^{\Gamma} 21^{\Gamma} 2 \omega_{3} 21^{\Gamma} \in \Lambda_{C} .
$$

It follows by induction on $n$ that $\left\langle a^{n}, b^{n}\right\rangle \in C-\operatorname{Access}\left(\mathscr{D}_{n}\right)$.
We consider a sequence of linear formulas constructed from the program $\Pi$, as in the Proof of Theorem 15; recall that these formulas do not contain disjunctions or equalities. Let $\phi(x, y)$ be a formula of the sequence (with free variables $x, y$ ) with $M$ variables, such that: (i) $\phi\left(a^{n}, b^{n}\right)$ holds in $\mathscr{D}_{n}$; and (ii) for any database $\mathscr{D}$, if $\phi(a, b)$ holds in $\mathscr{D}$ then $\langle a, b\rangle \in C-\operatorname{Access}(\mathscr{D})$. We will show that $M \geqslant n$; the result follows-from Theorem 15-since $\mathscr{D}_{n}$ is a simple path of length $(2 \Gamma+5)\left(3^{n}-1\right)$.

We consider the structure $\mathscr{H}_{\phi}=\left(H, \operatorname{arc}_{1}, \operatorname{arc}_{2}\right)$; we will show that there is a row of $\mathscr{H}_{\phi}$ with at least $n$ elements.

Arguing as in the Proof of Theorem 32, we show that $\mathscr{H}_{\phi}$ has at least one substructure isomorphic to $\mathscr{D}_{n}$. To avoid additional notation, we assume (with no loss of generality) that $\mathscr{D}_{n}$ is a substructure of $\mathscr{H}_{\phi}$.

Lemma. There is some substructure of $\mathscr{H}_{\phi}$ isomorphic to $\mathscr{D}_{n}$ such that: for each $i$, there is a row of $\mathscr{H}_{\phi}$ which intersects the paths of $\mathscr{D}_{i+1}$ from $a^{i+1}$ to $c_{1}$; from $c_{1}^{\prime}$ to $\chi$; from $\chi^{\prime}$ to $c_{2}$; and from $c_{2}^{\prime}$ to $b^{i+1}$.

Proof of Lemma. Assume it is false; we will derive a contradiction.
Consider an arbitrary copy of $\mathscr{D}_{n}$ in $\mathscr{H}_{\phi}$. There is some $i$ such that the statement of the Lemma is violated. By Lemmas 29 and 30, there is a row $V_{k_{0}}$ which "separates" two of the paths in the statement - say, without loss of generality, the path from $a^{i+1}$ to $c_{1}$ is separated from the path from $c_{2}^{\prime}$ to $b^{i+1}$. More precisely, this means that if the row $V_{k}$ of $\mathscr{H}_{\phi}$ intersects the path from $a^{i+1}$ to $c_{1}$, we have $k<k_{0}$; and if $V_{k}$ intersects the path from $c_{2}^{\prime}$ to $b^{i+1}$, we have $k_{0} \leqslant k$.

By a straightforward application of the "pumping" technique of [AC89, ACY95] (see [ACY95], Lemma 4.8) to the part of $\mathscr{H}_{\phi}$ consisting of the rows $V_{k}$ with $k<k_{0}$, we can show the following. If $\Gamma$ is sufficiently large with respect to the program, there is some formula $\phi^{\prime}$ of the sequence, such that $\mathscr{H}_{\phi^{\prime}}$ has the following property: $\mathscr{H}_{\phi^{\prime}}$ contains a path from $a^{i+1}$ to $b^{i+1}$, which is obtained by repeating-an arbitrary number of times - certain subpaths (see [ACY95], Definition 4.7) of the path from $a^{i+1}$ to $b^{i+1}$ contained in $\mathscr{H}_{\phi}$. Specifically, by the choice of $k_{0}$, the paths we repeat include some subpaths of the path from $a^{i+1}$ to $c_{1}$; and include no subpath of the path from $c_{2}^{\prime}$ to $b^{i+1}$. It is not hard to see (since the repetition can be done an arbitrary number of times) that $\mathscr{H}_{\phi^{\prime}}$ can be chosen so that the labels of the path from $a^{i+1}$ to $b^{i+1}$ in $\mathscr{H}_{\phi^{\prime}}$ do not correspond to a word in $\Lambda_{C}$.

By repeating this argument for each copy of $\mathscr{D}_{n}$ in $\mathscr{H}_{\phi}$, we obtain a structure $\mathscr{H}_{\psi}$, where $\psi(x, y)$ is some formula of the sequence; and if $e_{x}, e_{y}$ are the elements corresponding to the free occurences of $x, y$ in $\psi$ : there is no path from $e_{x}$ to $e_{y}$ in $\mathscr{H}_{\psi}$ with labels corresponding to a word in $\Lambda_{C}$. However, $\psi\left(e_{x}, e_{y}\right)$ holds in $\mathscr{H}_{\psi}$ (Lemma 31), so $\left\langle e_{x}, e_{y}\right\rangle \in C-\operatorname{Access}\left(\mathscr{H}_{\psi}\right)$; this is a contradiction.

Claim. Suppose $\mathscr{D}_{n}$ is a substructure of $\mathscr{H}_{\phi}$ as in the Lemma. Then, for each $i \geqslant 0$, there is a row of $\mathscr{H}_{\phi}$ which contains at least $i$ elements of $D_{i}$.

Proof of Claim. We argue by induction on $i$; the base case is obvious.
$\mathscr{D}_{i+1}$ is a simple path connecting the elements $a^{i+1}, b^{i+1}$. It contains three disjoint copies of $\mathscr{D}_{i}$; the $l$ th copy is a simple path $\mathscr{D}_{i, l}$ connecting the elements $a_{l}^{i}, b_{l}^{i}$.

Let $l=1,2,3$. Apply the inductive hypothesis to $\mathscr{D}_{i, l}$, to obtain a row $V_{k_{l}}$ of $\mathscr{H}_{\phi}$ containing at least $i$ elements of $\mathscr{D}_{i, l}$.

If any two of the indices $k_{1}, k_{2}, k_{3}$ coincide, there is a single row of $\mathscr{H}_{\phi}$ which contains at least $2 i$ elements of $\mathscr{D}_{i+1}$.

If $k_{1}, k_{2}, k_{3}$ are all different, assume (with no loss of generality) that $k_{1}<k_{2}<k_{3}$.
Let $V_{k_{0}}$ be a row of $\mathscr{H}_{\phi}$ as in the Lemma, intersecting the path of $\mathscr{D}_{i+1}$ from $a^{i+1}$ to $c_{1}$ at $A$; the path from $c_{1}^{\prime}$ to $\chi$ at $A^{\prime}$; the path from $\chi^{\prime}$ to $c_{2}$ at $B$; and the path from $c_{2}^{\prime}$ to $b^{i+1}$ at $B^{\prime}$.

If $k_{0} \leqslant k_{2}$, consider the path of $\mathscr{D}_{i+1}$ connecting $B$ to $B^{\prime}$. Note that this path contains no elements of $\mathscr{D}_{i, 2}$. Since it contains $\mathscr{D}_{i, 3}$, it intersects the row $V_{k_{3}}$; and by Lemma 29, it also intersects the row $V_{k_{2}}$. So $V_{k_{2}}$ contains at least one element of $\mathscr{D}_{i+1}$ which is not in $\mathscr{D}_{i, 2}$, i.e., $V_{k_{2}}$ contains at least $i+1$ elements of $\mathscr{D}_{i+1}$.

If $k_{0} \geqslant k_{2}$, consider the path of $\mathscr{D}_{i+1}$ connecting $A$ to $b_{1}^{i}$. Note that this path contains no elements of $\mathscr{D}_{i, 2}$. Since it contains $\mathscr{D}_{i, 1}$, it intersects the row $V_{k_{1}}$; and by Lemma 29, it also intersects the row $V_{k_{2}}$. So $V_{k_{2}}$ contains at least one element of $\mathscr{D}_{i+1}$ which is not in $\mathscr{D}_{i, 2}$, i.e., $V_{k_{2}}$ contains at least $i+1$ elements of $\mathscr{D}_{i+1}$.

It follows from the Claim that there is a row of $\mathscr{H}_{\phi}$ with at least $n$ elements.
The results in this Section can be extended to $\operatorname{Datalog}(\neq)$; and furthermore to $\operatorname{Datalog}(\neq, \neg)$, which in addition allows negation on EDB-predicates.

## 6. DISCUSSION AND OPEN PROBLEMS

We have shown that there are natural notions of complexity associated with Datalog programs. It seems worthwhile to examine more general logics (with various notions of fixpoint) from the same perspective. We feel that our concept of a linear formula might prove useful for such investigations.

We note that Immerman has also considered relationships between the number of variables of first-order formulas and computational complexity [CFI92, EI95, Imm81, Imm91]. His results are incomparable to ours, as the formulas he considers are more general, and the lower bounds he obtains are smaller.

Parallel evaluation of Datalog programs has been studied wrto communication costs [LV98]. It is interesting to see if an expressibility approach would also be useful in this case.

Our results immediately suggest several concrete questions. In particular, we believe that Theorems 24, 32 and 43 can be strengthened as follows.

Conjecture 47. Every Datalog $(\neq)$ program that expresses $K$-Paths has derivation dag size $\Omega\left(n^{K}\right)$.

Conjecture 48. Every sequence of linear formulas that expresses Access has $\Omega(n)$ variables.

Conjecture 49. Every Datalog program that expresses $K$-Equals has derivation dag size $\Omega\left(n^{K}\right)$.

The following is a conjectured nonuniform analogue of Theorem 43.
Conjecture 50. Every sequence of linear formulas that expresses $K$-Equals has depth $\Omega\left(n^{K}\right)$.

## REFERENCES

[AHV95] S. Abiteboul, R. Hull, and V. Vianu, "Foundations of Databases," Addison-Wesley, Reading, MA, 1995.
[A97] F. Afrati, Bounded arity Datalog $(\neq)$ queries on graphs, J. Comput. System Sci. 55 (1997), 210-228.
[AC89] F. Afrati and S. Cosmadakis, Expressiveness of restricted recursive queries, in "Proc. 21st ACM Symp. on Theory of Computing," pp. 113-126, 1989.
[ACY95] F. Afrati, S. Cosmadakis, and M. Yannakakis, On Datalog vs. polynomial time, J. Comput. System Sci. 51 (1995), 177-196.
[AF99] F. Afrati and E. Foustoucos, Datalog program transformations and persistency numbers, manuscript, 1999.
[AG94] M. Ajtai and Y. Gurevich, Datalog vs first-order logic, J. Comput. System Sci. 49 (1994), 562-588.
[AP93] F. Afrati and C. H. Papadimitriou, The parallel complexity of simple logic programs, J. Assoc. Comput. Mach. 40 (1993).
[AU79] A. V. Aho and J. D. Ullman, Universality of data retrieval languages, in "Proc. 6th ACM Symp. on Principles of Programming Languages," pp. 110-117, 1979.
[BKBR90] C. Beeri, P. Kanellakis, F. Bancilhon, and R. Ramakrishnan, Bounds on the propagation of selection into logic programs, J. Comput. System Sci. 41 (1990), 157-180.
[BMSU86] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman, Magic sets and other strange ways to implement logic programs, in "Proc. 5th ACM Symp. on Principles of Database Systems," pp. 1-15, 1986.
[BR91] C. Beeri and R. Ramakrishnan, On the power of magic, J. Logic Program. 10 (1991).
[CFI92] Y. Cai, M. Furer, and N. Immerman, An optimal lower bound on the number of variables for graph identification, Combinatorica 12 (1992).
[Eh61] A. Ehrenfeucht, An application of games to the completeness problem for formalized theories, Fund. Math. 49 (1961), 129-141.
[EI95] K. Etessami and N. Immerman, Tree canonization and transitive closure, in "Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science," 1995.
[Fr54] R. Fraissé, Sur quelques classifications des systèmes de relations, Publ. Sci. Univ. Alg. Sér. A 1 (1954), 35-182.
[Gai82] H. Gaifman, On local and nonlocal properties, in "Logic Coloquium '81" (J. Stern, Ed.), pp. 105-135, North Holland, Amsterdam, 1982.
[GJ79] M. Garey and D. Johnson, "Computers and Intractability: A Guide to the Theory of NPCompleteness," Freeman, San Francisco, 1979.
[Imm81] N. Immerman, Number of quantifiers is better than number of tape cells, J. Comput. System Sci. 22 (1981), 384-406.
[Imm87] N. Immerman, Languages that capture complexity classes, SIAM J. Comput. 16 (1987), 760-778.
[Imm91] N. Immerman, $\operatorname{DSPACE}\left(n^{k}\right)=\operatorname{VAR}(k+1)$, in "Proceedings of the 6th Annual Conference on Structure in Complexity Theory," 1991.
[KV95] Ph. G. Kolaitis and M. Y. Vardi, On the expressive power of Datalog: tools and a case study, J. Comput. System Sci. 51 (1995), 110-134.
[LM89] V.S. Lakshmanan and A. O. Mendelzon, Inductive pebble games and the expressive power of Datalog, in "Proc. 8th ACM Symp. on Principles of Database Systems," pp. 301-310, 1989.
[LV98] S. Lifschitz and V. Vianu, A probabilistic view of Datalog parallelization, Theoret. Comput. Sci. 190 (1998), 211-239.
[SZ88] D. Sacca and C. Zaniolo, The generalized counting method for recursive logic queries, Theoret. Comput. Sci. 62 (1988).
[Ul189] J. D. Ullman, "Database and Knowledge-Base Systems," Vols. I and II, Computer Science Press, New York, 1989.
[UvG88] J. D. Ullman and A. Van Gelder, Parallel complexity of logical query programs, Algorithmica 3 (1988), 5-42.


[^0]:    ${ }^{1}$ Not considered before, so far as we know.

[^1]:    ${ }^{2}$ The quantifier depth of a formula is the maximum number of quantifiers on a path for the root to a leaf, in a tree representation of the formula.

[^2]:    ${ }^{4}$ Existential formulas without negation.
    ${ }^{5}$ Depending on the program, but not depending on the size of the database.

[^3]:    ${ }^{7}$ That is, $\tau$ is obtained from $T$-up to isomorphism-by equating some values.

[^4]:    ${ }^{8}$ Noted by F. Afrati.

[^5]:    ${ }^{9}$ Linear lower bounds can be shown easily, by constructing databases where every element must be examined by the program.

