Undecidability and intractability results concerning Datalog programs and their persistency numbers

STAVROS COSMADAKIS
University of Patras
EUGENIE FOUSTOUCOS
Athens University of Economics and Business
and
ANASTASIOS SIDIROPOULOS
Massachusetts Institute of Technology

The relation between Datalog programs and homomorphism problems and, between Datalog programs and bounded treewidth structures has been recognized for some time and given much attention recently. Additionally, the essential role of persistent variables (in program expansions) for solving several relevant problems has also started to be observed. In [Afrati et al. 2005] the general notion of program persistencies was refined into four notions (two syntactical ones and two semantical ones) and the interrelationship between these four *persistency numbers* was studied. In the present paper (1) we prove undecidability results concerning the semantical notions of persistency number–modulo equivalence, of persistency number and of characteristic integer, (2) we exhibit new classes of programs for which boundedness is undecidable and (3) we prove intractabiltity results concerning the syntactical notions of weak persistency number and of weak characteristic integer.

Categories and Subject Descriptors: D.1.6 [**Programming Techniques**]: Logic Programming; F.1.1 [**Computation by abstract devices**]: Models of Computation—*Automata*; *Bounded-action devices*; *Computability theory* ; F.1.3 [**Computation by abstract devices**]: Complexity Measures and Classes—*Reducibility*; F.4.2 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Decision Problems*; H.2.3 [**Database Management**]: Languages—*Query Languages*

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: Datalog, bounded treewidth hypergraphs, Persistent variables, Persistency numbers, Boundedness, Undecidability, Intractability

Authors' addresses: Stavros Cosmadakis, University of Patras, 26500 Rio, Patras, Greece, email: scosmada@cti.gr.
Eugénie Foustoucos (Corresponding Author), MPLA, Department of Mathematics, National and Capodistrian University of Athens, Panepistimiopolis, 15784 Athens, Greece, and Department of Computer Science, Athens University of Economics and Business, 10434 Athens, Greece, email: aflaw@otenet.gr, eugenie@aueb.gr. Research partially supported by K. Karathéodory Basic Research Program no. 2997 of University of Patras, 2002 and by MPLA (Graduate Program in Logic, Algorithms and Computation), National and Capodistrian University of Athens.
Anastasios Sidiropoulos, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, email: tasos@mit.edu. Research partially supported by K. Karathéodory Basic Research Program no. 2997 of University of Patras, 2002.

1. INTRODUCTION

Datalog programs have been investigated extensively in the last two decades and several authors have observed that, in order to derive stronger results, a more thorough and fine investigation of their structure is needed [Cosmadakis et al. 1988; Cosmadakis 1989; Afrati and Cosmadakis 1989; Afrati 1997]. Specifically it has been observed that several issues may depend on the arity of recursive predicates and moreover on the number of occurrences of persistent variables (variables that appear both in the head of a rule and in recursive predicates in the body of that rule). Persistent variables (or their absence) appear to be important for decidability results, for instance the boundedness problem is proved undecidable in the general case [Gaifman et al. 1987] whereas when the arity of the recursive predicates is equal to one it becomes decidable [Cosmadakis et al. 1988]; note that when the recursive predicates have arity one, any persistent variable can be eliminated in a relatively straightforward way [Cosmadakis et al. 1988].

Persistent variables also appear to play an important role in expressive power issues: it is known that there exists a hierarchy of Datalog programs w.r.t. their arity [Afrati and Cosmadakis 1989]; programs of larger arity are strictly more expressive. There exists also such a hierarchy w.r.t. their "persistency number" [Afrati 1997]; the definition of this number involves semantical notions and not only syntactic [Afrati 1997].

The persistencies (appearing in program expansions) have an impact on the kind of models that a given program can have: the basic idea is that the more persistencies we need (in an essential way) for writing a program the more constraints are imposed on the models of the program, specifically on the structure of their local neighbourhoods.

Indeed every model of the program is a homomorphic image of one or more program expansion(s); through expansions, models are closely related to the syntax of the program in the following way: if model M is the homomorphic image of expansion e (via homomorphism h) then any two elements a and b of M are "close" to each other if their inverse images through h are variables x and y belonging to the same rule body; for instance when M is a path then a, b are consecutive nodes if there exists a binary predicate symbol E such that the atom $E(x, y)$ belongs to some rule body, for x inverse image of a and y inverse image of b.

A typical program having paths as models is the following:
$P(x, y) \leftarrow E(x, z), P(z, y)$
$P(x, y) \leftarrow E(x, y),$
with y being the variable that persists in expansions of any length (thus we say that the weak persistency number of the program is 1).

Clearly that program accepts any path, either simple or not: indeed every expansion can be mapped in a straightforward way (i.e. by a 1-1 and onto homomorphism) to a simple path, and in that way we obtain simple paths as models; furthermore, since the program is inequality-free, every homomorphic image of any of its models d is also a model (accepted by the same expansion as the one that accepts d, but via

a different homomorphism which will map distinct variables to the same element): such a model will be a non-simple path, and in that way we obtain non-simple paths as models.

Suppose now that we add $z \neq x$ to the first rule which becomes $P(x, y) \leftarrow E(x, z), P(z, y), z \neq x$. Every inequality $u \neq v$ present in some rule body used in a given expansion will prevent that expansion to accept a path through a mapping that associates the same element of the domain to both u and v. This slightly restricts the set of models, by eliminating paths that contain self-loops (in any of their nodes except the final one); every other kind of non-simple paths is allowed like for instance the path (a,b,c,d,e,c); the inequality $z \neq x$ concerns only adjacent variables in the expansion i.e. variables that belong to the same rule body; therefore its impact is local and this is due to the fact that none of the variables involved (i.e. x and z) is persistent. If now, instead of adding $z \neq x$, we add $z \neq y$ then the first rule becomes $P(x, y) \leftarrow E(x, z), P(z, y), z \neq y$. This imposes the following restriction on the set of models: a path $(x, z, z_1, ..., z_n, y)$ where $y = z$ or $y = z_i$ for $i = 1, ..., n$ will not be allowed (for instance the path (a,b,c,d,e,c) is not allowed anymore while the path (a,a,c,d,e,f) is now allowed). The restriction imposed is more severe now since a larger subset of non-simple paths is excluded from the set of models of the program; the reason is that the impact of the inequality $z \neq y$ is not local to a rule body anymore, because the variable y is persistent and thus the inequality propagates (in expansions) to variables arbitrarily far from each other.

The important result connecting persistencies and expressibility is Theorem 4.2 in [Afrati 1997]; which states that the H-subgraph homeomorphism query cannot be expressed by any Datalog program with inequalities which has persistency number less that $m-1$, where m is the number of edges of the digraph H. It is important to understand that, despite a very technical proof, that result is based on the simple aforementioned idea that persistencies have an impact on the kind of models that a program can have. We believe that it will be worth revisiting the proof of that Theorem 4.2 which we expect to become simpler provided that we take proper advantage of the new knowledge about persistencies given by the present paper and also by the previous ToCL paper [Afrati et al. 2005].

Persistent variables appear to be important for query evaluation and optimization techniques: it is well-known indeed that a Datalog program of size n with IDB predicates of arity at most k, can be evaluated in time $O(n^k)$; a substantial improvement to that bound seems unlikely, on complexity-theoretic grounds. On the other hand, there exist natural queries (among the H-subgraph homeomorphism queries) which need persistency number at least k-1 and at the same time seem to require $\Omega(n^k)$ time [Afrati 1997].

The concept of persistencies is implicit in [Cosmadakis et al. 1988],[Vardi 1988] and explicit in [Cosmadakis 1989]; syntactical definitions of persistency equal to zero are given in [Gaifman et al. 1987; 1993; Cosmadakis 1989]. The notion of persistencies is deeper studied and formalized in [Afrati et al. 2005] by means of four distinct persistency numbers. In the present paper we prove undecidability and intractability results concerning the persistency numbers; we believe these results will help to further our knowledge about the role persistent variables play in issues of expressibility, decidability and complexity of Datalog programs. Moreover we hope

that the negative results we have obtained will help future research on persistencies to be oriented towards more realistic directions (for instance development of ad hoc algorithms for increasing or decreasing the number of persistent variables of specific classes of programs).

In order to explain the various persistency numbers, we first illustrate how program's semantics are captured by the essential notion of expansion.

Let π_1 be the following program which expresses the transitive closure query via the intensional predicate T (it is the program that we have already considered and which has paths as models):

$r_1 : T(x,y) \leftarrow E(x,y)$

$r_2 : T(x,y) \leftarrow E(x,z), T(z,y)$

and let $e_0 = E(x,y)$ (obtained from rule r_1), $e_1 = \exists z(E(x,z) \wedge E(z,y))$ (obtained by unwinding rule r_2 with rule r_1) and $e_2 = \exists z \exists z_1(E(x,z) \wedge E(z,z_1) \wedge E(z_1,y))$ (obtained by unwinding twice rule r_2 and ending up with rule r_1) be three T-expansions of π_1 for the goal $T(x,y)$.

Consider the database $\mathcal{D}$ with domain $D=\{a,b,c,d\}$ and relation $\{(a,b),(b,c),(c,d)\}$. The facts $T(a,b)$, $T(a,c)$ and $T(a,d)$ are among the facts that can be deduced from the database $\mathcal{D}$ using program π_1; for each of these facts, we show below that their property of been deducible from $\mathcal{D}$ using π_1, can be formalized in terms of the existence of homomorphisms between distinguished databases; a database $\mathcal{D}$ is called distinguished whenever we mark some of its elements (for instance a_1,...,a_n) as distinguished, and it is then denoted by $(\mathcal{D}, a_1, ..., a_n)$.

a. The aforementioned deducibility of $T(a,b)$ (from $\mathcal{D}$) using π_1, is asserted by the existence of a T-expansion e of π_1 such that there exists a homomorphism from e (viewed as a distinguished database) to the distinguished database $(\mathcal{D}, a, b)$. Among the expansions of π_1, only one has the previous property, it is expansion e_0 formalized as the distinguished database $(\mathcal{C}_0, x, y)$ where $\mathcal{C}_0$ is the database with domain $C_0 = \{x,y\}$ and relation $\{(x,y)\}$. The homomorphism h_0 from the expansion $e_0 = (\mathcal{C}_0, x, y)$ of π_1 to $(\mathcal{D}, a, b)$ is such that $h_0(x) = a$, $h_0(y) = b$.

b. The deducibility of $T(a,c)$ (from $\mathcal{D}$) using π_1, is asserted by the existence of a T-expansion e of π_1 such that there exists a homomorphism from e (viewed as a distinguished database) to the distinguished database $(\mathcal{D}, a, c)$. Among the expansions of π_1, only one has the previous property, it is expansion e_1 formalized as the distinguished database $(\mathcal{C}_1, x, y)$ where $\mathcal{C}_1$ has domain $C_1 = \{x,y,z\}$ and relation $\{(x,z),(z,y)\}$. The homomorphism h_1 from the expansion $e_1 = (\mathcal{C}_1, x, y)$ of π_1 to $(\mathcal{D}, a, c)$ is such that $h_1(x) = a$, $h_1(z) = b$, $h_1(y) = c$.

c. The deducibility of $T(a,d)$ (from $\mathcal{D}$) using π_1, is asserted by the existence of a T-expansion e of π_1 such that there exists a homomorphism from e to the distinguished database $(\mathcal{D}, a, d)$. Among the expansions of π_1, only one has the previous property, it is expansion e_2 formalized as the distinguished database $(\mathcal{C}_2, x, y)$ where $\mathcal{C}_2$ has domain $C_2 = \{x,y,z,z_1\}$ and relation $\{(x,z),(z,z_1),(z_1,y)\}$. The homomorphism h_2 from the expansion $e_2 = (\mathcal{C}_2, x, y)$ of π_1 to $(\mathcal{D}, a, d)$ is such that $h_2(x) = a$, $h_2(z) = b$, $h_2(z_1) = c$, $h_2(y) = d$.

We now describe the four persistency numbers introduced in [Afrati et al. 2005] on some examples, starting with program π_1. Let us consider the rules of π_1 and count the number of variables that persist together from the head of some rule

to some intensional atom of the rule's body; we see that there is only one such variable (the variable y) in r_2 and no such variable at all in r_1; we say that the *syntactic persistency number* of program π_1 is 1. If we now consider program's expansions and count the variables that persist together in an unbounded number of expansions' rule bodies (or bubbles) we can determine the weak persistency number of program π_1; here the only way to built T-expansions of unbounded length is by repeatedly using rule r_2; the only variable which appears in every rule body of such an expansion is the variable y; this means that the *weak T-persistency number* of π_1 is 1. The *T-persistency number* of a program is a semantical notion and it somehow corresponds to the weak T-persistency number, but estimated not over the whole set of program's T-expansions, but over any set of T-expansions which is sufficient to express the semantics of the program (and which gives the lowest possible value for the T-persistency number). Here every T-expansion is needed to define the semantics of (π_1, T), therefore the T-persistency number of π_1 is equal to its weak T-persistency number i.e. it is 1.

We now come to a deeper semantical notion which is invariant under program equivalence: the *T-persistency number–modulo equivalence* of a program π, evaluated over all programs that are equivalent to π w.r.t. predicate T. The *T-persistency number–modulo equivalence* of π_1 is the minimum of the T-persistency numbers of all programs π expressing the transitive closure query via their common predicate T; it follows from our previous analysis that this number is at most equal to 1. It is in fact equal to 0, because there is another program π_0 consisting of the following rules:

$r_1^0 :\ T(x,y) \leftarrow E(x,y)$
$r_2^0 :\ T(x,y) \leftarrow E(x,z), E(z,y).$
$r_3^0 :\ T(x,y) \leftarrow E(x,z_1), T(z_1,z_2), E(z_2,y)$

such that (π_0, T) expresses the transitive closure query too and π_0 has no persistent variables at all (thus the T-persistency number of π_0 is 0).

It is not hard to understand that, for any program π and any IDB predicate symbol P of π, the syntactic P-persistency number a, the weak P-persistency number b, the P-persistency number c and the P-persistency number–modulo equivalence d of π, satisfy the inequality $arity(P) \geq a \geq b \geq c \geq d$.

The structure of the paper is the following. In section 2 we recall some basic preliminaries about Datalog and about expansions as bounded treewidth hypergraphs with persistencies. In section 3 we define the various persistency numbers (the most important are the weak persistency number and the persistency number–modulo equivalence). In section 4.1, we prove our undecidability result concerning the persistency number–modulo equivalence i.e. we prove that there is no algorithm to decide for any program whether its persistency number–modulo equivalence has a given value (and we derive, as corollaries, undecidability results for the persistency number and for the characteristic integer) and in section 4.2 we exhibit new classes of programs for which boundedness is undecidable (namely the class of programs of any fixed syntactic persistency number and the class of programs of any fixed weak persistency number). In section 5 we give our intractability results (summarized in Table I) concerning the weak persistency number and the weak characteristic integer. More precisely, we prove that given a program π, a predicate P of π and an

integer m, the problem of determining whether π has weak P-persistency number $\geq m$ is PSPACE-complete for linear Datalog programs, either normal or general; we prove that the problem is PSPACE-hard for normal non-linear Datalog programs and that it is APSPACE-complete for general non-linear Datalog programs. We derive similar results for the problem of determining whether, given a program π, a predicate P of π and an integer L, there exists a P-persistent set of length $\geq L$ and of size $m+1$ (m is the weak P-persistency number of π).

2. PRELIMINARIES: DATALOG PROGRAMS/QUERIES/EXPANSIONS

A *database* over domain D is a finite relational structure $\mathcal{D} = (D, r_1, \ldots, r_n)$, where D is a finite set and each r_i is a relation over D [Ullman 1988; Abiteboul et al. 1995]. The sequence $(\alpha_1, ..., \alpha_n)$ of arities of the r_i's is the *type* of the database. The database $\mathcal{D} = (D, r_1, \ldots, r_n)$ has signature $(R_1, ..., R_n)$ where for $i = 1, ..., n$ R_i is a predicate symbol of arity α_i, naming relation r_i. A *distinguished database* is a tuple $\mathcal{D}^* = (\mathcal{D}, a_1, ..., a_n)$ where $\mathcal{D}$ is a database and $(a_1, ..., a_n)$ is a tuple of elements of the domain D of $\mathcal{D}$. Let $\mathcal{D}^* = ((D, r_1, \ldots, r_m), a_1, ..., a_n)$ and $\mathcal{D}'^* = ((D', r'_1, \ldots, r'_m), a'_1, ..., a'_n)$ be two distinguished databases such that r_i has the same arity as r'_i for $i = 1, ..., m$; a *homomorphism* from $\mathcal{D}^*$ to $\mathcal{D}'^*$, is a total function $h : D \to D'$ such that (1) $h(a_i) = a'_i$ for $i = 1, ..., n$ and (2) if $(a_1, \ldots, a_{m_i}) \in r_i$, then $(h(a_1), \ldots, h(a_{m_i})) \in r'_i$. The composition of two homomorphisms is a homomorphism.

A *Datalog program* is a collection of rules of the form $\iota_0 \leftarrow \iota_1, \ldots, \iota_\kappa$ where ι_0 is the *head*, and $\iota_1, \ldots, \iota_\kappa$ form the *body* of the rule. Each ι_i is an expression of the form $R(t_1, \ldots, t_m)$, where the t_i's are terms i.e. either variables or constant symbols (we say constants for short) and R is a predicate symbol (we say predicate for short): R is either an *extensional database* (EDB) predicate naming one of the database relations, or an *intensional database* (IDB) predicate which is defined by the program; IDB predicates are exactly the ones appearing in the heads of rules [Ullman 1988; Abiteboul et al. 1995]. If R is an EDB (resp. IDB) predicate, then $R(t_1, \ldots, t_m)$ is an EDB (resp. IDB) *atom*. The set of EDB predicates is the *signature* of the input database. A *recursive* rule is a rule with at least one IDB predicate in its body; otherwise it is an *initialization* rule. The *dependency graph* of a program π is a directed graph with vertices the IDB predicates of π such that there exists an (unlabelled) edge from P to P' whenever π has a rule with head predicate P and with an IDB atom of predicate P' in its body. A program π is *recursive* if there exists a cycle in its dependency graph. A program is *linear* if each rule has at most one IDB atom in its body. Given a Datalog program π and a goal predicate P we may, starting with an atom over P, unwind the recursive rules of π to some finite depth (ending up with initialization rules), to obtain a *P-expansion* of π.

In the literature an expansion is viewed as a first-order formula over $\exists, \wedge$ and precisely as a conjunction of extensional atoms with a set of distinguished variables and with the remaining variables being existentially quantified (see examples in the Introduction).

We introduce a new formal (inductive) definition of expansions that will be convenient both for defining expansions as distinguished databases (see subsection 2.1)

and for defining them as hypergraphs (see subsection 2.2); notice that on this hypergraph definition is based the definition of *persistent sets of expansions*, a notion central in our work.

2.1 Expansions as distinguished databases. Queries and the notion of acceptance

We now give our new formal (inductive) definition of expansions as pairs of the form $(P(\vec{x}), \mathcal{D})$ where $P(\vec{x})$ is called the *head part* and $\mathcal{D}$ is called the *database part*: the head part $P(\vec{x})$ gives the distinguished variables $\vec{x}$ of the expansion and tells us that it is a P-expansion, while the database part $\mathcal{D}$ corresponds to the extensional atoms of the expansion.

DEFINITION 2.1. (**Expansions**) - If $P(\vec{x}) \leftarrow E_1(\vec{y}_1), ..., E_k(\vec{y}_k)$ is an initialization rule then $(P(\vec{x}), \{E_1(\vec{y}_1), ..., E_k(\vec{y}_k)\})$ is an expansion.
- If $(P(\vec{x}), \mathcal{D})$ is an expansion and σ is a substitution then $\sigma((P(\vec{x}), \mathcal{D}))$ is an expansion.
- If $P(\vec{x}) \leftarrow E_1(\vec{y}_1), ..., E_k(\vec{y}_k), P_1(\vec{z}_1), ..., P_l(\vec{z}_l)$ is a recursive rule with EDB predicates $E_1, ..., E_k$, $(P_1(\vec{z}_1),\mathcal{D}_1)$, ...,$(P_l(\vec{z}_l),\mathcal{D}_l)$ are expansions, and $Var(\mathcal{D}_i) \cap Var(\mathcal{D}_j) \subseteq Var(\vec{z}_i) \cap Var(\vec{z}_j)$ for each $i \neq j$, then $(P(\vec{x}), \{E_1(\vec{y}_1), ..., E_k(\vec{y}_k)\} \cup \mathcal{D}_1 \cup ... \cup \mathcal{D}_l)$ is also an expansion.

A natural question that arises is whether the first-order formula represented by a given expansion is satisfied in the structure represented by a given database: that logical notion of satisfiability is captured by the notion of P-acceptance which is expressed in terms of homomorphisms between distinguished databases (see definition 2.2), provided that we view expansions as distinguished databases; indeed any expansion $(P(t_1, ..., t_n), \mathcal{D})$ can be viewed as the distinguished database $(\mathcal{D}', t_1, ..., t_n)$ where $\mathcal{D}' = (D, r_1, \ldots, r_p)$, the domain D is equal to the set of terms occurring in $\mathcal{D}$ and, for $i = 1, ..., p$, $(s_1, ..., s_{m_i}) \in r_i$ if and only if $R_i(s_1, ..., s_{m_i}) \in \mathcal{D}$; by abuse of notation we always write $\mathcal{D}$ instead of $\mathcal{D}'$. Examples are given in the Introduction.

DEFINITION 2.2. The distinguished database $\mathcal{D}^*$ is P-accepted by the P-expansion $e = (P(t_1, ..., t_n), \mathcal{C})$ of π if and only if there exists a homomorphism h from e - viewed as the distinguished database $(\mathcal{C}, t_1, ..., t_n)$ - to $\mathcal{D}^*$, such that for every constant b, $h(b) = b$.
$\mathcal{D}^*$ is P-accepted by program π if and only if $\mathcal{D}^*$ is P-accepted by some P-expansion of π.

DEFINITION 2.3. A *query* is a function Q from databases (of some fixed type) to relations of fixed arity such that the image of a database with domain D is a relation on D. [1]
Let P be an IDB predicate of program π. The pair (π, P) defines a query $Q_{\pi,P}$ as follows: for every database $\mathcal{D}$, $Q_{\pi,P}(\mathcal{D})$ is the set of tuples $(d_1, d_2, \ldots, d_m)$ such that the distinguished database $(\mathcal{D}, d_1, ..., d_m)$ is P-accepted by π.

[1] A query Q has to be *generic*, i.e. invariant under renamings of the domain; this means that, for every database $\mathcal{D} = (D, r_1, \ldots, r_m)$ and for every domain D' such that there exists a one-to-one correspondence $i : D \longrightarrow D'$, $Q(i(\mathcal{D})) = i(Q(\mathcal{D}))$ where $i(\mathcal{D})$ denotes the database $(i(D), i(r_1), \ldots, i(r_m))$: if r is a relation on D then $i(r)$ is the relation on $D' = i(D)$ such that $(b_1, ..., b_t) \in r$ if and only if $(i(b_1), ..., i(b_t)) \in i(r)$.

DEFINITION 2.4. 1. A recursive program π is *bounded w.r.t. its IDB predicate* P if there exists an integer K such that every distinguished database accepted by some P-expansion of π is accepted by some P-expansion (of π) of depth $< K$; this property is called *predicate boundedness*.
2. A recursive program π is *bounded* if it is bounded w.r.t. all its IDB predicates; this property is called *program boundedness*.
3. Programs π_1 and π_2 are P-equivalent iff every P-expansion of program π_1 is P-accepted by π_2 and vice-versa (P occurs in both π_1 and π_2).

Obviously a non recursive program is bounded. Suppose now that program π is recursive and that, in the dependency graph of π, there is no path starting from P and reaching some cycle; then obviously, π is bounded w.r.t. P. It has been proved that (1) a program is bounded w.r.t. P if and only if it is P-equivalent to a non recursive program and that (2) a program π is bounded w.r.t. all its IDB predicates P if and only if, for each IDB predicate P in π, program π is P-equivalent to a non recursive program. Notice that the decidability of predicate boundedness implies the decidability of program boundedness; but the converse does not hold [Marcinkowski 1999].

2.2 Skeleton trees and tree-decompositions of expansions viewed as hypergraphs. The central notion of persistent set.

DEFINITION 2.5. (**Skeleton Tree**) A tree $\mathcal{T}$ is a *skeleton tree* associated to program π if $\mathcal{T}$ satisfies the following: 1) its nodes are labeled with rules of π and 2) a node N with label r has exactly n sons N_1,..., N_n with respective labels r_1,...,r_n if and only if rule r has exactly n occurrences $P_1(\vec{x}_1)$,...,$P_n(\vec{x}_n)$ of IDB atoms in its body and, for $i = 1, ..., n$, P_i is the head predicate of r_i.

The depth of a skeleton tree is the maximal depth of its nodes (where the root has depth 0 and a node N has depth $n + 1$ if its father has depth n).

With each expansion e of π, we can associate (in a straightforward way) a skeleton tree of π. Such skeleton trees have all their leaves labelled with initialization rules. The depth of an expansion is the depth of its associated skeleton tree, if that tree is unique; the skeleton tree associated to an expansion is unique in most cases. However, there are some "pathological" cases where the skeleton tree associated with an expansion is not unique: consider for instance the program consisting of the following two rules $P(x, y) \leftarrow P(y, x)$ and $P(x, y) \leftarrow E(x, y)$, and consider the expansion $(P(x, y), E(x, y))$ which has infinitely many associated skeleton trees; in such cases we associate to the expansion e the (unique) skeleton tree of minimal depth. The skeleton tree associated to an expansion e is called the *skeleton tree of* e.

Every expansion $e = (P(\vec{x}), \mathcal{D})$ can be viewed as a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{HE})$ such that the vertices in $\mathcal{V}$ are the terms (variables or constants) that appear in e and each hyperedge $(t^1, ..., t^n)$ with label E in $\mathcal{HE}$ is defined from the EDB atom $E(t^1, ...t^n)$ in $\mathcal{D}$; every element of $\vec{x}$ is a distinguished vertex of the hypergraph $\mathcal{H}$.

Any Datalog expansion e, viewed as a hypergraph $\mathcal{H}$, has a tree-like structure given by the tree-decomposition of $\mathcal{H}$; we give below the definition of a hypergraph's tree-decomposition, which is analogous to the - well-known from graph theory - notion of a graph's tree-decomposition. Another notion, also important to our work,

is that of bounded tree-width hypergraphs which we define below, still by analogy to the notion of bounded tree-width graphs. As stated in [Diestel 2006] (notes at the end of chapter 12), the graph-theoretical notions of tree-decomposition and tree-width were first introduced (under different names) by R. Halin [Halin 1976]. Robertson and Seymour reintroduced these two concepts, fundamental in their work on Graph Minors, which has been appearing in the Journal of Combinatorial Theory, Series B, since 1983. Since then, these concepts have been elaborated on and used in many research papers ([Courcelle 1990; Gottlob et al. 2001] etc.).

DEFINITION 2.6. Let $\mathcal{H} = (\mathcal{V}, \mathcal{HE})$ be a hypergraph (with set of vertices $\mathcal{V}$ and set of hyperedges $\mathcal{HE}$). A *tree-decomposition* of $\mathcal{H}$ is a pair $(\mathcal{T}, f)$, where $\mathcal{T}$ is a tree (with set of nodes $\mathcal{N}$) and $f : \mathcal{N} \rightarrow \mathcal{P}(\mathcal{V})$ maps every node i of $\mathcal{T}$ to a set $f(i)$ - called *bubble* - of vertices of $\mathcal{H}$ such that
(1) $\mathcal{V} = \bigcup\{f(i) \mid i \in \mathcal{N}\}$,
(2) every hyperedge of $\mathcal{H}$ has its vertices in some set $f(i)$,
(3) if $v \in f(i) \cap f(j)$, then $v \in f(k)$ for every k belonging to the unique path in $\mathcal{T}$ linking i to j.

DEFINITION 2.7. The width of a tree-decomposition $(\mathcal{T}, f)$ of a hypergraph $\mathcal{H}$ is the maximal cardinality of its bubbles minus one, i.e. $max\{|f(i)| \mid i \in \mathcal{N}\} - 1$. The tree-width of $\mathcal{H}$ is the minimum width of a tree-decomposition of $\mathcal{H}$. A family $\mathcal{F}$ of hypergraphs is of bounded tree-width if there exists a constant k such that every hypergraph $\mathcal{H} \in \mathcal{F}$ has tree-width at most k [2].

The skeleton trees associated to a program π provide natural tree-decompositions of the expansions of π (this fact, implicit in [Kolaitis and Vardi 1998], has been proved in [Afrati et al. 2005]); each of these tree-decompositions has width at most equal to m-1 where m is the maximum number of distinct terms occurring in a rule of π; the family of expansions of π is therefore a family of hypergraphs of bounded treewidth.

We come now to the central notion of *persistent set* of an expansion; this notion comes from the structure of expansions as hypergraphs as explained in the following definition.

DEFINITION 2.8. (**Persistent Set**) Let $\mathcal{H}$ be a hypergraph and $(\mathcal{T}, f)$ be a tree-decomposition of $\mathcal{H}$. For every subtree L of $\mathcal{T}$ with $l > 1$ bubbles (i.e. nodes of $\mathcal{T}$) b_1, b_2,..., b_l such that $A = b_1 \cap b_2 \cap ... \cap b_l \neq \emptyset$, the set A is called *persistent set of length* l; the cardinality of A is called *size* of A. The elements of a persistent set are called *persistencies*. If $\mathcal{H}$ has a persistent set A of size n and length l, we say that $\mathcal{H}$ has n persistencies of length l. If the subtree L is reduced to a branch b with l bubbles b_1, b_2,..., b_l (where b_i is the father of b_{i+1}) then the corresponding persistent set of length l is called *linear persistent set on branch* (b_1, b_2,..., b_l).

3. PERSISTENCY NUMBERS OF A DATALOG PROGRAM

Four persistency numbers, concerning various "levels" of presence of persistent terms, have been defined in [Afrati et al. 2005] (see examples in the Introduction):

[2]When a hypergraph $\mathcal{H}$ belongs to a family of hypergraphs of bounded tree-width, we say that $\mathcal{H}$ is a hypergraph of bounded tree-width.

two of these numbers are syntactical, the simpler one is the *syntactical persistency number* defined in a straightforward manner from the syntactical form of the program, while the *weak persistency number* is a more elaborate notion since it concerns the presence of persistent [3] terms in the set of all expansions of the program. The other two numbers correspond to semantical notions: the *persistency number* concerns programs while the *persistency number–modulo equivalence* concerns queries and is therefore connected to deeper issues (indeed, among the four numbers, it is the only persistency number which is invariant up to program equivalence). We also remind that it has been proved in [Afrati et al. 2005] that any program of persistency number m is equivalent to some program of weak persistency number m. It follows that the *weak persistency number* and the *persistency number–modulo equivalence* are the two most interesting of these numbers: the present paper essentially deals with them.

DEFINITION 3.1. Let π be a program and let P be an IDB predicate of π.
1. Program π has *weak P-persistency number* m if m is the minimum integer satisfying the following: there exists an integer k such that for every P-expansion e of π having $> m$ persistencies of length l, it is true that $l < k$.
Program π has *weak persistency number* m if m is the maximum among those integers n satisfying the following: there exists an IDB predicate P of π such that n is the weak P-persistency number of π.
2. The *P-persistency number–modulo equivalence* of a program π is the minimum integer m such that π is P-equivalent to a program of weak P-persistency number m.
We also say that the *persistency number–modulo equivalence* of a query $\mathcal{Q}$ is the P-persistency number–modulo equivalence of any pair (π, P) defining $\mathcal{Q}$.

We define below the notions of persistency number and of characteristic integer, needed at the end of section 4.1. and also the notion of weak characteristic integer, needed in section 5.2.

DEFINITION 3.2. 1. Program π has *P-persistency number* m if m is the minimum integer satisfying the following: there exists an integer k such that for every database D which is P-accepted by π, there exists a P-expansion e that accepts D such that if e has $> m$ persistencies of length l then $l < k$.
Program π has *persistency number* m if m is the maximum among those integers n satisfying the following: there exists an IDB predicate P of π such that n is the P-persistency number of π.
2. We call *P-characteristic integer* (resp. *weak P-characteristic integer*) of π the minimum integer satisfying the requirements of the integer k in the definition of the P-persistency number (resp. weak P-persistency number) of π.
We call *characteristic integer* (resp. *weak characteristic integer*) of π the maximum among those integers l satisfying the following: there exists an IDB predicate P of π such that l is the P-characteristic integer (resp. weak P-characteristic integer) of π.[4]

[3] See definition 2.8.
[4] The characteristic integer is introduced in the present paper while the weak characteristic in-

All previous definitions are related to persistent sets of expansions and refer to their natural (via skeleton trees) tree-decompositions of bounded tree-width. One might use other tree-decompositions of expansions for defining persistency numbers and these numbers would be the same, as long as the tree- decompositions used are of bounded tree-width. Notice however that the specific choice of the tree-decomposition (of bounded tree-width) may affect the characteristic integer.

4. UNDECIDABILITY RESULTS

4.1 Undecidability of the persistency number–modulo equivalence and related results

In this subsection, we mainly study the decision problem "does a given program have persistency number–modulo equivalence equal to m, for any fixed $m > 0$"; we prove that this problem is undecidable by reducing to it some undecidable problem concerning context-free grammars (CFG), namely the problem of deciding whether the language produced from a CFG G is equal to the set of all strings of terminal symbols of G [Hopcroft and Ullman 1979]. At the end of the subsection we prove (using essentially the same reduction) two more undecidability results, one concerning the persistency number and the other concerning the characteristic integer. The proof is based on three lemmas stated below. The first lemma gives the construction, for any integer $n > 0$, of a program π_n of persistency number–modulo equivalence equal to n, which defines a query subsuming the transitive closure query.

LEMMA 4.1. Let $n > 0$ and consider the program π_n consisting of the following rules (where E and A are EDB predicates):
$r_1 : T_n(x, y, z_1, ..., z_n) \leftarrow E(x, x'), T_n(x', y', z_1, ..., z_n), E(y', y), A(x, x', z_1, ..., z_n)$
$r_2 : T_n(x, y, z_1, ..., z_n) \leftarrow E(x, y), A(x, y, z_1, ..., z_n)$.
$r_3 : G_n(x, y) \leftarrow T_n(x, y, z_1, ..., z_n)$.
The T_n-persistency number–modulo equivalence and the G_n-persistency number–modulo equivalence of π_n are both equal to n.

PROOF 4.2. First consider the predicate T_n of arity $n+2$. Clearly the weak T_n-persistency number of π_n is equal to n because the distinguished variables $z_1, ..., z_n$ form a persistent set of maximal size and of length that grows - in an unbounded way - as the length of (skeleton trees of) expansions grows. Consider now any program π which is T_n-equivalent to π_n. If there is a homomorphism from a T_n-expansion e_n of π_n to a T_n-expansion e of π, then the E-paths in e and e_n must have same length (this also holds when there is a homomorphism from e to e_n); therefore the length of T_n-expansions in π grows as grows the length of their respective homomorphic inverse images which are T_n-expansions of π_n. Any homomorphism from some T_n-expansion e of π to some T_n-expansion e_n of π_n maps the distinguished variables $u_1, ..., u_n$ of the predicate T_n of π on the distinguished variables $z_1, ..., z_n$ of the predicate T_n of π_n (and conversely, any homomorphism from e_n to e maps $z_1, ..., z_n$ to $u_1, ..., u_n$). Therefore the set $\{u_1, ..., u_n\}$ of distinguished variables is necessarily persistent (of unbounded length) in T_n-expansions of π, which means that π (which

teger was introduced in [Afrati et al. 2005] under the name of characteristic integer w.r.t.-weak-persistency-number.

is an arbitrary program T_n-equivalent to π_n) has weak T_n-persistency number at least n. Since π_n has weak T_n-persistency number exactly equal to n, we have proved that the T_n-persistency number–modulo equivalence of π_n is equal to n.

Consider now the predicate G_n of arity 2. Obviously there is no difference between predicates T_n and G_n as far as the weak persistency number is concerned, thus the weak G_n-persistency number of π_n is n. When considering programs that are G_n-equivalent to π_n we must be carefull since $\{z_1, ..., z_n\}$ is not a set of distinguished variables anymore; now it is the atoms $A(x^{(i)}, x^{(i+1)}, z_1, ..., z_n)$ and $A(y^{(i)}, y^{(i+1)}, z_1, ..., z_n)$ that force the image $\{u_1, ..., u_n\}$ of $\{z_1, ..., z_n\}$ to be persistent for unbounded length. Thus the G_n-persistency number–modulo equivalence of π_n is equal to n. □

The second lemma constructs from every CFG G, a Datalog program π_G with set of IDB predicates containing predicates P and Q such that (i) $(\pi_G, P) \subseteq (\pi_G, Q)$ (i.e. every distinguished database accepted by a P-expansion of π_G is also accepted by a Q-expansion of π_G) and (ii) it is undecidable, for given G, if $(\pi_G, P) = (\pi_G, Q)$ (i.e. $(\pi_G, P) \subseteq (\pi_G, Q)$ and $(\pi_G, Q) \subseteq (\pi_G, P)$).

LEMMA 4.3. Let G be a context-free grammar (CFG) with set of terminal (resp. nonterminal) symbols T (resp. N) and let $L(G)$ be the language produced by G. From G we can construct a Datalog program π_G of persistency number 1 containing among its IDB predicates, the predicates P and Q and such that:
(1) $(\pi_G, P) \subseteq (\pi_G, Q)$ and
(2) $L(G) = T^*$ iff $(\pi_G, P) = (\pi_G, Q)$.

PROOF 4.4. For every terminal symbol $t \in T$ of G, we create an EDB predicate E_t. For each nonterminal symbol $u \in N$ of G we create an IDB predicate P_u. For each production $a \to a_1 a_2 ... a_n$ of G we create rule $P_a(x, y) \leftarrow T_{a_1}(x, z_1), T_{a_2}(z_1, z_2), ..., T_{a_n}(z_{n-1}, y)$ where T_{a_i} denotes the IDB P_{a_i} if a_i is a nonterminal symbol of G and T_{a_i} denotes the EDB E_{a_i} if a_i is a terminal symbol of G. The IDB predicate Q is defined by rules $Q(x, y) \leftarrow E_t(x, y)$, $t \in T$ and $Q(x, y) \leftarrow E_t(x, z), Q(z, y)$, $t \in T$.
The IDB predicate P is defined by rule $P(x, y) \leftarrow P_S(x, y)$ if S is the initial symbol of G.
It is easy to see that there exists a one-to-one correspondence between (a) the set T^* of all strings formed by terminal symbols and (b) the set of Q-expansions of π_G; and that there exists a one-to-one correspondence between (a) the language $L(G)$ produced by G and (b) the set of P-expansions of π_G. This proves that (1) $(\pi_G, P) \subseteq (\pi_G, Q)$ (since $L(G) \subseteq T^*$) and (2) $(\pi_G, P) = (\pi_G, Q)$ if and only if $L(G) = T^*$. Without loss of generality we can assume that G does not contain productions of the form $a \to b$ where $a, b \in N$, therefore π_G does not contain rules of the form $T_a(x, y) \leftarrow T_b(x, y)$ and its persistency number is equal to 1. □

The third lemma gives the reduction which proves the undecidability of the persistency number–modulo equivalence.

LEMMA 4.5. Let G be a context-free grammar as in lemma 4.3. From G and for every integer $m > 0$ we can construct a Datalog program $\pi_{G,m}$ with goal predicate K such that the following conditions are equivalent:

(i) $L(G) = T^*$
(ii) $\pi_{G,m}$ has K-persistency number–modulo equivalence equal to m.

PROOF 4.6. Program $\pi_{G,m}$ consists of two strata. The first stratum is program π_G defined in lemma 4.3. The second stratum is a program (with goal predicate K) that uses three new EDB predicates E, A, B, and uses also as EDB predicates, the IDB predicates P and Q of the first stratum in order to compute "(P, E)-paths" and "(Q, E)-paths".

The rules that compute (P, E)-paths are essentially the rules of program π_n of lemma 4.1 for $n = m$, with two slight modifications (1. the use of both E-atoms and P-atoms and 2. the adjunction of the EDB atom $B(z_m, u)$), that do not affect the persistency number–modulo equivalence which is equal to m :
$K(x, y) \leftarrow K_P(x, y)$
$K_P(x, y) \leftarrow T_m(x, y, z_1, ..., z_m)$
$T_m(x, y, z_1, ..., z_m) \leftarrow E(x, x'), T_m(x', y', z_1, .., z_m), E(y', y), A(x, x', z_1, .., z_m), B(z_m, u)$
$T_m(x, y, z_1, ..., z_m) \leftarrow P(x, y), A(x, y, z_1, ..., z_m), B(z_m, u)$.

The rules that compute (Q, E)-paths are essentially the rules of program π_n of lemma 4.1 for $n = m + 1$, with two slight modifications (1. both E-atoms and P-atoms are used, 2. the EDB atom $A(x, y, z_1, ..., z_{m+1})$ is replaced by the IDB atom $S(x, y, z_1, ..., z_{m+1})$ defined by a unique rule which is an initialization rule) that do not affect the persistency number–modulo equivalence, which is equal to $m + 1$:
$K(x, y) \leftarrow K_Q(x, y)$
$K_Q(x, y) \leftarrow T_{m+1}(x, y, z_1, ..., z_{m+1})$
$T_{m+1}(x, y, z_1, ..., z_{m+1}) \leftarrow E(x, x'), T_{m+1}(x', y', z_1, ..., z_{m+1}), E(y', y), S(x, x', z_1, ..., z_{m+1})$
$T_{m+1}(x, y, z_1, ..., z_{m+1}) \leftarrow Q(x, y), S(x, y, z_1, ..., z_{m+1})$
$S(x, y, z_1, ..., z_{m+1}) \leftarrow A(x, y, z_1, ...z_m), B(z_m, z_{m+1})$

We can notice that the persistency number–modulo equivalence of $\pi_{G,m}$ is given by its second stratum which has weak persistency number equal to $m + 1$ (because its weak K_P-persistency number is equal to m and its weak K_Q-persistency number is equal to $m+1$), while the first stratum (i.e. program π_G of lemma 4.3) has weak persistency number equal to 1 and does not contribute to the weak persistency number of $\pi_{G,m}$ and it doesn't contribute either to its persistency number–modulo equivalence. We can therefore focus our attention on the second stratum.

Considering P as an EDB predicate, we see that K_P-expansions have either the form $(K_P(x, y), \{P(x, y), A(x, y, z_1, ..., z_m), B(z_m, u)\})$ or the form $(K_P(x, y), \{E(x, x_1), ..., E(x_{n-1}, x_n), P(x_n, y_n), E(y_n, y_{n-1}), ..., E(y_1, y), A(x, x_1, z_1, ..., z_m), ..., A(x_{n-1}, x_n, z_1, ..., z_m), A(x_n, y_n, z_1, ..., z_m), B(z_m, u_1), ..., B(z_m, u_{n+1})\})$ for $n \geq 1$. K_Q-expansions have exactly the same form as K_P-expansions, provided that we replace P by Q and we replace the atoms $B(z_m, u_1), ..., B(z_m, u_{n+1})$ by the unique atom $B(z_m, z_{m+1})$. It is not hard to see that the following equivalence holds: every K_Q-expansion is accepted by a K_P-expansion (of $\pi_{G,m}$) iff every Q-expansion is accepted by a P-expansion (of π_G) that is $L(G) = T^*$, according to lemma 4.3.

We come now to the proof of the equivalence of the following conditions:
(i) $L(G) = T^*$
(ii) $\pi_{G,m}$ has K-persistency number–modulo equivalence equal to m.

First we show that (i) $\Longrightarrow$ (ii).
Suppose that $L(G) = T^*$, this means that every K_Q-expansion is accepted by a K_P-expansion, i.e. every (Q, E)-path can be obtained as a (P, E)-path. Thus the K_Q-expansions are unnecessary for computing the query $(\pi_{G,m}, K)$ which means that the K-persistency number–modulo equivalence of $\pi_{G,m}$ is equal to its K_P-persistency number–modulo equivalence, thus equal to m.

We now show that (ii) $\Longrightarrow$ (i) i.e we show that if $L(G) \neq T^*$ then $\pi_{G,m}$ has K-persistency number–modulo equivalence different from m and therefore equal to $m + 1$.
We suppose that $L(G) \neq T^*$, this means that $T^* \not\subset L(G)$ (since $L(G) \subset T^*$); therefore there exists at least one element of T^* which is not in $L(G)$, i.e. there exists a Q-path which is not a P-path. Combining that Q-path with E-paths of arbitrary length we obtain an infinite set S of K_Q-expansions, each of which is not accepted by any K_P-expansion. Let e_{m+1} be an expansion in the set S. Consider any program π which is Q-equivalent to $\pi_{G,m}$. If there is a homomorphism from the expansion e_{m+1} of $\pi_{G,m}$ to a K_Q-expansion e of π, then the E-paths in e and $e_m + 1$ must have the same length. Therefore there is a set R of K_Q-expansions in π (the homomorphic images of the expansions of $\pi_{G,m}$ in S) whose length grows with the length of their respective homomorphic inverse images in the (infinite) set S. Arguing as in the proof of lemma 4.1, we can show that the image of the elements $\{z_1, ..., z_{m+1}\}$ of the expansions in S must be persistent for unbounded length in the expansions of π that belong to the set R. It follows that the weak K-persistency number of π is at least $m+1$. Therefore the K-persistency number–modulo equivalence of $\pi_{G,m}$ is equal to $m + 1$. □

THEOREM 4.7. The decision problem: "does a given program have P-persistency number–modulo equivalence equal to m?" is undecidable, for any fixed $m > 0$.

PROOF 4.8. Suppose that for each fixed $m > 0$, there is an algorithm which decides, for every query (π, P), if π has P-persistency number–modulo equivalence equal to m. Then we would be able to decide whether the persistency number–modulo equivalence of the query $(\pi_{G,m}, K)$ of lemma 4.5 is equal to m or not, thus we would be able to decide, for every CFG G, whether $L(G) = T^*$ which is known to be undecidable [Hopcroft and Ullman 1979]. □

¿From the undecidability of the P-persistency number–modulo equivalence we can derive analogous undecidability results for the P-persistency number and for the P-characteristic integer, both defined in definition 3.2; these results are given in the following corollary.

COROLLARY 4.9. 1. The decision problem: "does a given program have P-persistency number equal to m?" is undecidable, for any fixed $m > 0$.
2. The decision problem: "does a given program have P-characteristic integer equal to m?" is undecidable, for any fixed $m \geq 3$.

PROOF 4.10. 1. Same reduction as for theorem 4.7, noticing that program $\pi_{G,m}$, $m > 0$, of lemma 4.5 has K-persistency number equal to m iff $L(G) = T^*$. Clearly if $\pi_{G,m}$ has K-persistency number equal to m then the K_Q-expansions are not needed for defining the query $(\pi_{G,m}, K)$, which means that the Q-paths can be obtained

as P-paths i.e. $(\pi_G, P) = (\pi_G, Q)$ which is equivalent to $L(G) = T^*$, according to lemma 4.3. Now in order to prove the converse, it is sufficient - according to lemma 4.5 - to prove that if program $\pi_{G,m}$ has K-persistency number–modulo equivalence equal to m then it has K-persistency number equal to m; the latter (i.e. the fact that $\pi_{G,m}$ has K-persistency number equal to m) is true since on the one hand the K-persistency number of $\pi_{G,m}$ is at least equal to m (it has been proved in [Afrati et al. 2005] that for any query (π, P), the P-persistency number–modulo equivalence of π is the minimum of the P-persistency numbers of programs P-equivalent to π) and on the other hand it is - by definition of the various persistency numbers - at most equal to its weak K-persistency number which is - by construction of $\pi_{G,m}$ - equal to m.

2. The proof is based on a stronger version of lemma 4.3: we construct a Datalog program π''_G by adding a new goal predicate and some "dummy" rules to program π_G of lemma 4.3 (after we have renamed its predicate P into P'); more precisely:

(a) we replace rule $P(x, y) \leftarrow P_S(x, y)$ of π_G with rules:

$P'(x, y) \leftarrow P_1(x, y)$

$P_1(x, y) \leftarrow P_2(x, y)$

...

$P_{m-4}(x, y) \leftarrow P_{m-3}(x, y)$

$P_{m-3}(x, y) \leftarrow P_S(x, y)$,

(b) we replace rules $Q(x, y) \leftarrow E_t(x, y)$ and $Q(x, y) \leftarrow E_t(x, z), Q(z, y)$ $(t \in T)$ of π_G with rules:

$Q(x, y) \leftarrow Q_1(x, y)$

$Q_1(x, y) \leftarrow Q_2(x, y)$

...

$Q_{m-2}(x, y) \leftarrow Q_{m-1}(x, y)$

$Q_{m-1}(x, y) \leftarrow E_t(x, y)$, $t \in T$

$Q_{m-1}(x, y) \leftarrow E_t(x, z), Q_{m-1}(z, y)$, $t \in T$.

Rules of π_G corresponding to productions of the CFG G, remain unchanged in π''_G.

(c) The new goal predicate P is defined by the new rules $P(x, y) \leftarrow P'(x, y)$ and $P(x, y) \leftarrow Q(x, y)$. It is not hard to see that the claims of lemma 4.3 not only remain true for program π''_G but are also enriched: (1) $(\pi''_G, P') \subseteq (\pi''_G, Q)$ and (2) the three following conditions are equivalent:

(i) $L(G) = T^*$

(ii) $(\pi''_G, P') = (\pi''_G, Q)$

(iii) the P-characteristic integer of π''_G is equal to m.

The proof of the equivalence (i) $\Leftrightarrow$ (iii) is based on the following remarks: (a) the adjunction of the above "dummy" rules has as effect that every persistent set which consists in strictly more than one (one is the persistency number of π_G and of $\pi_{G''}$) persistent variable, persists in $m - 1$ bubbles when the expansion is a P'-expansion and it persists in m bubbles when the expansion is a Q-expansion ($m \geq 3$ because, according to definition 2.8, any persistent set persists in at least 2 bubbles); therefore, if $L(G)$ is a strict subset of T^* (i.e. $(\pi''_G, P') \neq (\pi''_G, Q)$) then the P'-characteristic integer of program π''_G is equal to m while the Q-characteristic integer of π''_G is equal to $m + 1$; in that case the P-characteristic integer of π''_G is equal to $m + 1$, since both P'-expansions and Q-expansions are needed in order

to define the program semantics i.e. the query (π''_G, P). However, when $L(G) = T^*$ i.e. $(\pi''_G, P') = (\pi''_G, Q)$, the program semantics can be defined by using P'-expansions only; therefore the P-characteristic integer of π''_G becomes equal to its P'-characteristic integer which is equal to m, as we saw before.
Based on the above equivalence (i) $\Leftrightarrow$ (iii), and knowing that it is undecidable whether $L(G) = T^*$, we conclude that the decision problem: "does a given program have P-characteristic integer equal to m?" is undecidable, for any fixed $m \geq 3$. □

As far as theorems 4.7 and 4.9(1) are concerned, it is possible to prove the case $m = 0$ by a different technique using a (more involved technically) reduction based on directly encoding Turing machine computations.

4.2 New classes of programs for which boundedness is undecidable

LEMMA 4.11. For every integer $M \geq 1$, there exists an algorithm $\mathcal{B}_M$ which on every input program π produces an output program π'_M such that the following holds:
1. π is bounded if and only if π'_M is bounded,
2. π has weak persistency number $m \geq 0$ if and only if π'_M has weak persistency number $m + M$.

PROOF 4.12. 1. Fix the integer $M \geq 1$. For every input program π, algorithm $\mathcal{B}_M$ produces an output program π'_M: π'_M is obtained (1) by replacing every n-ary IDB predicate Q occurring in π with a new $n + M$-ary IDB predicate Q', (2) by adding a new M-ary EDB predicate A and (3) by choosing M new distinct variables $x_1, ..., x_M$ such that every recursive rule $r : Q_0(\vec{x}_0) \leftarrow Q_1(\vec{y}_1), ..., Q_n(\vec{y}_n), e_1, ..., e_s$ of π is replaced with the new recursive rule $r' : Q'_0(x_1, ..., x_M, \vec{x}_0) \leftarrow Q'_1(x_1, ..., x_M, \vec{y}_1)$, $..., Q'_n(x_1, ..., x_M, \vec{y}_n), e_1, ..., e_s, A(x_1, ..., x_M)$ and every initialization rule $r : Q'_0(\vec{x}_0)$ $\leftarrow e_1, ..., e_s$ of π is replaced with the new initialization rule $r' : Q'_0(x_1, ..., x_M, \vec{x}_0) \leftarrow$ $e_1, ..., e_s, A(x_1, ..., x_M)$.
For every database $\mathcal{D}$ (of domain dom) and for every vector $(a_1, ..., a_M)$ of distinct constants, we denote by $\mathcal{D}_{\{A,a_1,...,a_M\}}$ (of domain $dom_{\{a_1,...,a_M\}} = dom \bigcup \{a_1, ..., a_M\}$) the database obtained by enriching $\mathcal{D}$ with a new M-ary relation A which is true only on the M-tuple $(a_1, ..., a_M)$. It is not hard to see that, for any tuple $(b_1, ..., b_n)$ of dom, the distinguished database $\mathcal{D}^* = (\mathcal{D}, b_1, ..., b_n)$ is accepted by π iff the distinguished database $\mathcal{D}^*_{\{A,a_1,...,a_M\}} = (\mathcal{D}_{\{A,a_1,...,a_M\}}, a_1, ..., a_M, b_1, ..., b_n)$ is accepted by π'_M. Also e viewed as the distinguished database $(\mathcal{C}, b_{x_1}, ..., b_{x_n})$ is a P-expansion of π if and only if $e_{\{A,a_1,...,a_M\}}$ viewed as the distinguished database $(\mathcal{C}_{\{A,a_1,...,a_M\}}, a_1, ..., a_M, b_{x_1}, ..., b_{x_n})$ is a P-expansion of π'_M. Moreover it is not hard to see that if e^1 and e^2 are expansions of π and $e^1_{\{A,a_1,...,a_M\}}$ and $e^2_{\{A,a_1,...,a_M\}}$ are the corresponding expansions of π' then there is a homomorphism from e^1 to e^2 (viewed as distinguished databases) if and only if there is a homomorphism from $e^1_{\{A,a_1,...,a_M\}}$ to $e^2_{\{A,a_1,...,a_M\}}$ (viewed as distinguished databases): the "only if" direction is clear. To prove the "if" direction, let $h_{a_1,...,a_M}$ be a homomorphism from $e^1_{\{A,a_1,...,a_M\}}$ to $e^2_{\{A,a_1,...,a_M\}}$ such that, for $i = 1, ..., M$, $h_{a_1,...,a_M}(a_i) = a_i$ and let h be the restriction of $h_{a_1,...,a_M}$ on e^1. Consider any constant b of e^1 $(b \neq a_i,\ i = 1, ..., M)$ which is mapped by h on some constant a_i, $i = 1, ..., M$, of $e^2_{\{A,a_1,...,a_M\}}$; since h is a homomorphism, every such constant b is not connected

- via some extensional predicate E - to any other constant $c \neq a_i$, $i = 1, ..., M$, of $e^1_{\{A,a_1,...,a_M\}}$ (i.e b and c do not belong to a same tuple of the relation defined by E). Therefore we can map b on any constant of e^2 and construct, in such a way, a homomorphism h^* from e^1 to e^2; and this ends up the proof of the "if" direction. 2. Immediate from the construction of π'_M given at the beginning of the proof. □

THEOREM 4.13. For every fixed $M \geq 1$, boundedness is undecidable for programs that have weak persistency number M.

PROOF 4.14. The proof is based on the result that "program boundedness is undecidable for binary linear Datalog programs with a single IDB predicate" (theorem 2.3 in [Hillebrand et al. 1995] which is the journal version of [Hillebrand et al. 1991] and also in [Vardi 1988]). The undecidability reduction in [Hillebrand et al. 1995] constructs a program π_0 which has weak persistency number equal to 0 (this proves the theorem for $M = 0$ and has been proved in [Afrati et al. 2005]); on input π_0 and for every $M > 0$, algorithm $\mathcal{B}_M$ of lemma 4.11 produces a program π' of weak persistency number M. If we could decide if π' is bounded or not then we could decide if π_0 is bounded or not; but this is impossible (because π_0 is a binary linear Datalog program with a single IDB predicate and we know that program boundedness is undecidable for such programs). □

We recall the definition of the *syntactic P-persistency number* of a program, notion used in the following theorem. Program π has *syntactic P-persistency number* m if m is the maximum integer among those integers n satisfying the following: there exists in π a rule ρ defining predicate P such that n variables of $head(\rho)$ occur in some IDB atom of $body(\rho)$; program π has *syntactic persistency number* m if m is the maximum among those integers n satisfying the following: there exists an IDB predicate P of π such that n is the syntactic P-persistency number of π.

THEOREM 4.15. For every fixed $M \geq 1$, boundedness is undecidable for programs that have syntactic persistency number M.

PROOF 4.16. In [Afrati et al. 2005] (lemma 7.2) it has been proved that for every program π of weak persistency number M we can effectively construct a program π' equivalent to π and having syntactic persistency number M. Therefore, since boundedness is invariant under program equivalence, the following holds: if boundedness were decidable for programs of syntactic persistency number M then it would be decidable for programs of weak persistency number M. The result stated in theorem 4.13 completes the proof. □

The statements in theorems 4.13 and 4.15 are not only true for $M \geq 1$ but also for $M = 0$. The particular cases $M = 0$, $M = 2$ and $M = 3$ have been proved in [Afrati et al. 2005].

5. INTRACTABILITY RESULTS

Definition of the two problems (each of which has been proved to be decidable in [Afrati et al. 2005]):

Problem DATALOG PROGRAM WEAK PERSISTENCY NUMBER
Instance: Datalog program π, predicate P, integer m.
Question: Does π have weak P-persistency number at least m?

Datalog Program Weak Persistency Number	
Normal Linear	PSPACE-complete
Normal Non-Linear	PSPACE-hard
General Linear	PSPACE-complete
General Non-Linear	APSPACE-complete
Datalog Program Weak Characteristic Integer	
Normal Linear	PSPACE-complete
Normal Non-Linear	PSPACE-hard
General Linear	PSPACE-complete
General Non-Linear	APSPACE-complete

Table I. Summary of results.

Problem Datalog Program Weak Characteristic Integer
Instance: Datalog program π, predicate P, integer L.
Question: Does there exist a persistent set of size $m+1$, having length at least L, where m is the weak P-persistency number of π?

We consider two interesting cases for each one of these problems: (1) Datalog programs in normal form i.e., for every atom $Q(\vec{t})$ occurring in the program, if Q is an IDB predicate then $\vec{t}$ is a vector of distinct terms (either variables or constants) and (2) general, i.e. non normal, Datalog programs. The notion of *normal* programs extends the notion of *normal* programs that has been defined in [Afrati et al. 2005] for programs with no constants. For each one of these two cases, we consider the following two sub-cases: *linear* Datalog programs, and *non-linear* Datalog programs. All the presented results are summarized in Table I. [Hopcroft and Ullman 1979] and [Garey and Johnson 1979] are classical references for the concepts related to intractability, reductions, completeness and so on.

It is worth noticing that the complexity results of that section are related to complexity results for Datalog derivability (see lemma 5.9 and theorem 5.15). This is not surprising since it can be proved that the problem of deciding the weak persistency number is at least as hard as deciding, for a program P and a ground atom A, if A is derivable from P.

5.1 The problem DATALOG PROGRAM WEAK PERSISTENCY NUMBER

Theorem 5.1. The problem Normal Linear Datalog Program Weak Persistency Number is PSPACE-hard.

Proof 5.2. We reduce the problem Finite State Automata Intersection which is known to be PSPACE-hard to the problem Normal Linear Datalog Program Weak Persistency Number. That is, for every integer $n \geq 3$ and for every set of n non-deterministic automata $\mathcal{A} = \{A_0, ..., A_{n-1}\}$ over the same input alphabet Σ [5], we construct a linear program $\pi_{n,\mathcal{A}}$ in normal form and with goal

[5]Each automaton A_i has ϵ-transitions which allows us to assume that its final state is unique. W.l.o.g. we can assume that automata A_0,...,A_{n-1} have the same number of states; for $0 \leq i < n$ we denote by $K_i = \{q_i^0, q_i^1, \ldots, q_i^{|K|-1}\}$ the set of states of A_i where q_i^0 is the initial state and q_i^1 is the single accepting state.

predicate P, such that $\pi_{n,\mathcal{A}}$ has weak P-persistency number n if and only if there exists a string of Σ^* which is accepted by every automaton A_i of $\mathcal{A}$.

The IDB predicates of program $\pi_{n,\mathcal{A}}$ are the goal predicate P of arity n and predicates P_i, Q_i, $P_{a,i}$ and $Q_{a,i}$ of arity $n|K|$ ($0 \leq i < n$ and $a \in \Sigma$). Program $\pi_{n,\mathcal{A}}$ uses n constant symbols $c_0, c_1, \ldots, c_{n-1}$ and has several kinds of rules for $a, b \in \Sigma$:
(1) Rules $r_a : P(c_0, c_1, ..., c_{n-1}) \leftarrow P_{a,0}(\vec{y}_0, \vec{y}_1, ..., \vec{y}_{n-1})$ where $\vec{y}_i = (c_i, y_i^1, ..., y_i^{|K|-1})$.
(2) Rules r^{PQ}_{a,i,q_i^j,q_i^l} and r^{PQ}_{i,q_i^j,q_i^l} ($0 \leq i \leq n-1$) created for $j = l$ or for each sequence of empty transitions of A_i starting from q_i^j and terminating to q_i^l; rules r^{QP}_{a,i,q_i^j,q_i^l} ($0 \leq i < n-1$) created for each transition $q_i^j \xrightarrow{a} q_i^l$ of automaton A_i ($0 \leq j, l < |K|$):
$r^{PQ}_{a,i,q_i^j,q_i^l} : P_{a,i}(\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_i, \ldots, \vec{y}_{n-1}) \leftarrow Q_{a,i}(\vec{y}_0, \vec{y}_1, \ldots, \vec{u}_i, \ldots, \vec{y}_{n-1})$
$r^{PQ}_{i,q_i^j,q_i^l} : P_i(\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_i, \ldots, \vec{y}_{n-1}) \leftarrow Q_i(\vec{y}_0, \vec{y}_1, \ldots, \vec{u}_i, \ldots, \vec{y}_{n-1})$
$r^{QP}_{a,i,q_i^j,q_i^l} : Q_{a,i}(\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_i, \ldots, \vec{y}_{n-1}) \leftarrow P_{a,i+1}(\vec{y}_0, \vec{y}_1, \ldots, \vec{u}_i, \ldots, \vec{y}_{n-1})$
where $\vec{y}_i = (y_i^0, y_i^1, ..., y_i^{l-1}, y_i^l, y_i^{l+1}, ..., y_i^{|K|-1})$, $\vec{u}_i = (y_i^{'0}, y_i^{'1}, ..., y_i^{'l-1}, y_i^j, y_i^{'l+1}, ..., y_i^{'|K|-1})$ (if y_i^k is a variable then $y_i^{'k}$ is a new variable, and if y_i^k is a constant then $y_i^{'k} = y_i^k$).
(3) Rules $r^{QP}_{a,b} : Q_{a,n-1}(\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_{n-1}) \leftarrow P_{b,0}(\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_{n-1})$
(4) Rules $r'_a : Q_{a,n-1}(\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_{n-1}) \leftarrow P_0(\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_{n-1})$
(5) Rules $r^{QP}_i : Q_i(\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_{n-1}) \leftarrow P_{i+1}(\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_{n-1})$
(6) Rules $r^{QP}_{n-1} : Q_{n-1}(\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_i, \ldots, \vec{y}_{n-1}) \leftarrow P(y_0^1, y_1^1, \ldots, y_{n-1}^1)$
Rules $r^{QE}_{n-1} : Q_{n-1}(\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_i, \ldots, \vec{y}_{n-1}) \leftarrow E(y_0^1, y_1^1, \ldots, y_{n-1}^1)$

It is important to notice that in most of the above linear rules, the $n|K|$ arguments of the head are changed into the $n|K|$ arguments of the body according to the following pattern: (1) the $n|K|$ arguments are partitioned into n "blocks", each block representing an automaton and having K arguments, one argument per state, where in particular, the first (resp. second) position of an i-block denotes the initial (resp. final) state of A_i, (2) only one of the blocks is modified, (3) the modification of the block consists in renaming all its arguments, except one which (possibly) changes position: precisely a term moves from the $(j+1)^{th}$ position to the $(l+1)^{th}$ position of an i-block if and only if $q_i^j \xrightarrow{a} q_i^l$ is a transition of automaton A_i.

We prove the theorem by showing the equivalence of the following two propositions:
(i) $\{c_0, c_1, ..., c_{n-1}\}$ is the persistent set (of maximum size) occurring in P-expansions of arbitrary length (which means that the weak P-persistency number of the program is n)
(ii) there is a string of Σ^* which is accepted by every automaton A_i of $\mathcal{A}$.
(i) $\Longrightarrow$ (ii) Any P-expansion has the form $(P(c_0, ..., c_{n-1}), \mathcal{C})$ and a close look at rules' arguments shows that if there exists a persistent set A of maximal size then necessarily A is equal to the set of constants $\{c_0, c_1, ..., c_{n-1}\}$. The expansion necessarily starts with some rule r_a of group (1) which puts each constant c_i in the first position of the i-block; now $\{c_0, c_1, ..., c_{n-1}\}$ is persistent for arbitrary length iff rule r^{QP}_{n-1} of group (6) can be used an arbitrary number of times, producing - each time it is used - the atom $P(c_0, ..., c_{n-1})$ with which the expansion started; it is sufficient to show that r^{QP}_{n-1} is used once and it produces

$P(c_0, ..., c_{n-1})$: this is possible only if - at the time we choose to use r^{QP}_{n-1} - each constant c_i is in the second position of the i-block (in the head of the rule); to achieve that, the expansion must start (after rule r_a) with an a-sequence i.e. a sequence $r^{PQ}_{a,0,...}, r^{QP}_{a,0,...}, r^{PQ}_{a,1,...}, r^{QP}_{a,1,...}, ..., r^{PQ}_{a,j,...}, r^{QP}_{a,j,...}, ..., r^{PQ}_{a,n-1,...}$ of rules of group (2), at the end of which either (α) the constants are put in the second position of blocks attesting that symbol a has been accepted by the n automata or (β) rule $r^{QP}_{a,b}$ is used, followed by a b-sequence describing n transitions reading symbol b, one transition per automaton. At that point we repeat the same reasoning; eventually each constant c_i is in the second position of the i-block iff the string $a_1a_2...a_m$ has been accepted by the n automata, where the sequences of rules used in the expansion are first an a_1-sequence, then an a_2-sequence,...,eventually an a_m-sequence.
(ii) $\Longrightarrow$ (i): the proof is analogous to the proof of (i) $\Longrightarrow$ (ii) stated above. □

THEOREM 5.3. The following two problems are in NLINEARSPACE:
(1) The problem NORMAL LINEAR DATALOG PROGRAM WEAK PERSISTENCY NUMBER
(2) The problem GENERAL LINEAR DATALOG PROGRAM WEAK PERSISTENCY NUMBER.

PROOF 5.4. We can give a necessary and sufficient condition for the existence of a persistent set that occurs in an unbounded number of bubbles, by properly enriching the usual notion of dependency graph of a program π, as follows: for each tuple (P, P', r) such that there exists a rule $r : P(\vec{u}) \leftarrow ..P'(\vec{v}), ...$ of π, we create an edge $P \xrightarrow{r} P'$ and for each pair (P, r) such that r is an initialization rule with head over P we create an edge $P \xrightarrow{r} \circ$ where $\circ$ is a new symbol. If π has weak P_0-persistency number at least m, then there is a persistent set A of size m such that, for every integer l, there exists a P_0-expansion where A persists in l bubbles. Such a persistent set $A = \{x_1, ..., x_m\}$ exists if and only if (1) there is a cycle $T \xrightarrow{r_1} ... \xrightarrow{r_n} T$ in the enriched dependency graph of π where T is reachable from P_0, (2) there exist a partial T-expansion $e = (T(\vec{x}), \mathcal{C}, T(\vec{y}))$ [6] of π, and a sequence of indices u_1,...,u_m such that, for $i = 1, ..., m$, the element x_i of the persistent set A is the u_i^{th} term of both $\vec{x}$ and $\vec{y}$ (if we call $\vec{p}$ the subvector $(x_{u_1}, ..., x_{u_m})$ of $\vec{x}$ and we call $\vec{p}'$ the subvector $(y_{u_1}, ..., y_{u_m})$ of $\vec{y}$ then the condition (2) above says that $\vec{p} = \vec{p}'$ and thus $A = \{x_{u_1}, ..., x_{u_m}\} = \{y_{u_1}, ..., y_{u_m}\}$) and (3) from predicate T we can reach a predicate T' which appears in the head of some initialization rule (condition (3) assures the existence of T-expansions).

Therefore a non deterministic algorithm that can solve the problem NORMAL LINEAR DATALOG PROGRAM WEAK PERSISTENCY NUMBER guesses (1) an IDB predicate T of π, (2) a cycle $T \xrightarrow{r_1} ... \xrightarrow{r_n} T$ in the enriched dependency graph of π and (3) a set of m distinct terms, $m \leq arity(T)$. Such an algorithm needs to write down T, $\vec{p}$ and $\vec{p}'$ in order to test whether $\vec{p} = \vec{p}'$ (i.e. whether the m distinct terms constitute a persistent set); it therefore needs space linear w.r.t. the size of the program. □

[6] Expansions are defined from skeleton trees (recall definition 2.5) having all their leaves labelled with initialization rules while partial expansions are defined in an analogous way but from skeletons trees that have at least one leaf labelled with a recursive rule. (Partial) expansions of a linear program have their skeleton trees reduced to a branch.

THEOREM 5.5. The problem NORMAL LINEAR DATALOG PROGRAM WEAK PERSISTENCY NUMBER is PSPACE-complete.

PROOF 5.6. According to theorem 5.3 (1) the problem is in PSPACE (since NLINEARSPACE $\subset$ NPSPACE and NPSPACE=PSPACE); and according to theorem 5.1 it is PSPACE-hard. □

THEOREM 5.7. The problem NORMAL NON-LINEAR DATALOG PROGRAM WEAK PERSISTENCY NUMBER is PSPACE-hard.

PROOF 5.8. The class of linear programs is a subclass of the class of non-linear programs. Therefore the problem is at least as hard as the problem NORMAL LINEAR DATALOG PROGRAM WEAK PERSISTENCY NUMBER which is PSPACE-hard according to theorem 5.1. □

LEMMA 5.9. Given a Datalog program π and given an atom $P(\vec{u})$ over the IDB predicate P of π, we can decide in alternating linear space whether there exists a $P(\vec{u})$-expansion of π.

PROOF 5.10. Consider the following non deterministic algorithm which takes as inputs a program π and an atom $P(\vec{u})$ over the IDB predicate P of π:
1. Find an instance r of some initialization rule of π such that $P(\vec{u})$ is head of r;
2. if such r exists then stop and say "there exists $P(\vec{u})$-expansion",
3. otherwise find an instance r of some recursive rule of π such that $P(\vec{u})$ is head of r, and for every IDB atom $Q(\vec{t})$ of the body of r, check recursively whether there exists a $Q(\vec{t})$-expansion of π.

Recall that a computation, expressed by a tree, needs alternating linear space [Chandra et al. 1981] if and only if the computation of every branch of the tree needs deterministic linear space. The computation performed by the above algorithm can be represented as a tree; the computation on every branch of that tree consists in a sequence of atoms, starting with the atom $P(\vec{u})$.

To perform the computation along any branch, the algorithm has only to write down $P(\vec{u})$ as well as each pair of successive rule instances encountered; the algorithm does not need to write down rule instances, but it can codify them in the following way: if r' is an instance of program's rule r, the algorithm writes down (a) which terms of r have become equal in r' to a term of $\vec{u}$ (and to which one) and (b) among the remaning terms of r, which terms have been equated in r'. Therefore, in order to perform the computation along any branch of the tree, the algorithm uses space which is linear w.r.t. the size of the program. □

THEOREM 5.11. (1) The problem GENERAL NON-LINEAR DATALOG PROGRAM WEAK PERSISTENCY NUMBER is in ALINEARSPACE.
(2) The problem GENERAL NON-LINEAR DATALOG PROGRAM WEAK PERSISTENCY NUMBER is in EXPTIME.

PROOF 5.12. (1) The proof is based on the proof of theorem 5.3, but it also uses lemma 5.9. More precisely, there exists a persistent set $A = \{x_1, ..., x_m\}$ if and only if four conditions are satisfied: the three conditions (1), (2) [7], (3) in the

[7]Since programs are non-linear, condition (2) is now stated as "there exists a partial T-expansion

proof of theorem 5.3, plus a fourth condition: (4) there exist expansions $(I_1(\vec{y_1}), \mathcal{C}_1)$ ...,$(I_n(\vec{y_n}), \mathcal{C}_n)$.

Thus a non deterministic algorithm that can solve the problem must do - in space linear w.r.t. the size of the program - the three guesses corresponding to the three aforementioned conditions (according to the proof of theorem 5.3), but moreover it has to decide whether there exist an $I_1(\vec{y_1})$-expansion of π,..., an $I_n(\vec{y_n})$-expansion of π, and this can be done in alternating linear space according to lemma 5.9. The algorithm therefore needs space alternating linear w.r.t. the size of the program.
(2) Follows from (1) since ALINEARSPACE⊆APSPACE and APSPACE=EXPTIME. □

THEOREM 5.13. The problem GENERAL NON-LINEAR DATALOG PROGRAM WEAK PERSISTENCY NUMBER is APSPACE-complete, even for fixed weak persistency number, as long as it is at least 2.

PROOF 5.14. By theorem 5.11 the problem is in APSPACE; to show that it is APSPACE-hard, we will reduce to it the APSPACE-hard problem ALTERNATING POLYNOMIAL SPACE TURING MACHINE ACCEPTANCE. Let $M = (K, \Sigma, \delta, s, H)$ be a alternating Turing machine [Chandra et al. 1981], where K is a finite set of states, $\Sigma = \{0, 1, \sqcup, \triangleright\}$ is the alphabet (where $\sqcup$ denotes the blank and $\triangleright$ denotes the beginning of any string), $s \in K$ is the initial state, $H = \{q_{\text{accept}}, q_{\text{reject}}\} \subseteq K$ is the set of halting states, and $\delta : (K - H) \times \Sigma \to \mathcal{P}(K \times \Sigma \times \{\leftarrow, \rightarrow\})$ is the transition relation. Let p be a polynomial such that M on an input of length n, uses at most $p(n)$ cells of its tape. Let also $x = x_0, x_1, \ldots x_{|x|-1}$ be a string, with $x_1 = \triangleright$, and $x_i \in \{0, 1\}, 0 \leq i < n$. We will construct a non-linear Datalog program π_x, with a predicate P, and we will compute an integer m, such that π_x has weak persistency number at least m, iff M accepts input x, using at most $p(|x|)$ cells of its tape.

Let $N = |\Sigma| + p(|x|)$. We begin with a program π_x, having a single predicate P, of arity $|\Sigma|$. For each $q \in K$, and for each $i, 0 \leq i < p(|x|)$, we add the predicate $P_{q,i}$, of arity N. Next, for each $q \in K - H$, we add the following rules:

—If q is an existential state, then for each $q' \in K$, for each $a, a' \in \Sigma$, with $(q, a) \vdash_M (q', a', s)$, for some $s \in \{\leftarrow, \rightarrow\}$, let $l = -1$, if $s = \leftarrow$, and $l = 1$ otherwise. For each $i, 0 \leq i < p(|x|)$, we add the following rule:

$$P_{q,i}(y_0, y_1, y_\sqcup, y_\triangleright, z_1, \ldots, z_{i-1}, y_a, z_{i+1}, \ldots, z_{p(|x|)}) \leftarrow$$
$$P_{q',i+l}(y_0, y_1, y_\sqcup, y_\triangleright, z_1, \ldots, z_{i-1}, y_{a'}, z_{i+1}, \ldots, z_{p(|x|)})$$

—If q is a universal state, then for each $a \in \Sigma$, we construct the set $Q_{q,a} = \{(q_1, a_1), \ldots, (q_{n_{q,a}}, a_{n_{q,a}})\}$, containing all the tuples (q_j, a_j), such that $(q, a) \vdash_M (q_j, a_j, s_j)$ for some $s_j \in \{\leftarrow, \rightarrow\}$. For each $j, 1 \leq j \leq n_{q,a}$, let $l_j = -1$, if $s_j = \leftarrow$, and $l_j = 1$ otherwise. For each $i, 0 \leq i < p(|x|)$, we add the following rule:

$$P_{q,i}(y_0, y_1, y_\sqcup, y_\triangleright, z_1, \ldots, z_{i-1}, y_a, z_{i+1}, \ldots, z_{p(|x|)}) \leftarrow$$

$e = (T(\vec{x}), \mathcal{C}, \{T(\vec{y}), I_1(\vec{y_1}), ..., I_n(\vec{y_n}\}) ...$". $T(\vec{y})$, $I_1(\vec{y_1})$,...,$I_n(\vec{y_n})$ are the respective heads of the recursive rules labelling leaves in the skeleton tree of e.

$$P_{q_1,i+l_1}(y_0, y_1, y_\sqcup, y_\triangleright, z_1, \ldots, z_{i-1}, y_{a_1}, z_{i+1}, \ldots, z_{p(|x|)}),$$
$$P_{q_2,i+l_2}(y_0, y_1, y_\sqcup, y_\triangleright, z_1, \ldots, z_{i-1}, y_{a_2}, z_{i+1}, \ldots, z_{p(|x|)}), \ldots,$$
$$P_{q_{n_{q,a}},i+l_{n_{q,a}}}(y_0, y_1, y_\sqcup, y_\triangleright, z_1, \ldots, z_{i-1}, y_{a_{n_{q,a}}}, z_{i+1}, \ldots, z_{p(|x|)})$$

—Furthermore, for each $i, 0 \leq i < p(|x|)$, we add the rules

$$P_{q_{\text{accept}},i}(y_0, y_1, y_\sqcup, y_\triangleright, z_1, \ldots, z_{p(|x|)}) \leftarrow P(y_0, y_1, y_\sqcup, y_\triangleright)$$
$$P_{q_{\text{accept}},i}(y_0, y_1, y_\sqcup, y_\triangleright, z_1, \ldots, z_{p(|x|)}) \leftarrow E(y_0, y_1, y_\sqcup, y_\triangleright)$$
$$P(y_0, y_1, y_\sqcup, y_\triangleright) \leftarrow P_{s,0}(y_0, y_1, y_\sqcup, y_\triangleright, y_{x_0}, y_{x_1}, \ldots, y_{x_{|x|-1}}, y_\sqcup, \ldots, y_\sqcup).$$

For the constructed program π_x, it is easy to see that π_x has weak P-persistency number $|\Sigma| = 4$ if and only if M accepts x; indeed this is based on the following facts: (1) if A is a persistent set such that, for every n there exists a P-expansion in which A appears and has length at least n, then A has size at most 4, (2) M accepts x if and only if there exist $P(y_0, y_1, y_\sqcup, y_\triangleright)$-expansions (i.e. if and only if rules with head over predicate $P_{q_{\text{accept}},i}$ are reachable for some i) and (3) M does not accept x if and only if the weak P-persistency number is less than $4 = |\Sigma|$ (in that case it is equal to 0).

For every $m > 4$, we can slightly modify the previous reduction, by increasing the size of the alphabet Σ of the Turing machine in order to make $|\Sigma|$ equal to m; the arity of the constructed program π_x increases accordingly and π_x has weak P-persistency number $m = |\Sigma|$ if and only if M accepts x. To prove the theorem for $m = 2$ (resp. $m = 3$), it suffices to encode the 4 letters-alphabet Σ into a two (resp. three) letters alphabet Σ'. □

THEOREM 5.15. Given a general Datalog program π and an atom $P(\vec{x})$ over an IDB predicate of π, the problem of determining whether there exists a $P(\vec{x})$-expansion is APSPACE-complete.

PROOF 5.16. According to lemma 5.9 the problem is in ALINEARSPACE $\subset$ APSPACE. To show that it is APSPACE-hard, we reduce to it the APSPACE-hard problem ALTERNATING POLYNOMIAL SPACE TURING MACHINE ACCEPTANCE, using the same reduction as that in the proof of Theorem 5.13. □

THEOREM 5.17. The problem GENERAL LINEAR DATALOG PROGRAM WEAK PERSISTENCY NUMBER is PSPACE-complete, even for fixed weak persistency number, as long as it is at least 2.

PROOF 5.18. By Theorem 5.3 (2), the problem is in PSPACE; to show that it is PSPACE-hard, we will reduce to it the PSPACE-hard problem POLYNOMIAL SPACE TURING MACHINE ACCEPTANCE.

The proof is essentially the same as the proof of theorem 5.13 but now $M = (K, \Sigma, \delta, s, H)$ is a deterministic polynomial space Turing machine with transition function $\delta : (K - H) \times \Sigma \to K \times \Sigma \times \{\leftarrow, \rightarrow\}$. The new program π_x is similar to the program in the proof of 5.13 but it is linear since it does not contain the rules corresponding to universal states (while it conserves every other type of rule and especially the rules that were produced for existential states). □

5.2 The problem DATALOG PROGRAM WEAK CHARACTERISTIC INTEGER

The intractability results given here have proofs that are similar to the proofs in section 5.1.

THEOREM 5.19. (1) The problem NORMAL LINEAR DATALOG PROGRAM WEAK CHARACTERISTIC INTEGER is PSPACE-complete.
(2) The problem NORMAL NON-LINEAR DATALOG PROGRAM WEAK CHARACTERISTIC INTEGER is PSPACE-hard.

PROOF 5.20. (1) The problem is PSPACE-hard, even for fixed L: the proof is based on the reduction described in the proof of Theorem 5.1 (we have to add $L+1$ dummy rules similar to the dummy rules in the proof of corollary 4.9(2)). The problem is in PSPACE since we can prove that it is NLINEARSPACE: the proof is similar to the proof of theorem 5.3; a non deterministic algorithm that can solve the problem (i.e. check, for any normal linear program π of weak P-persistency number equal to m, whether there exists a persistent set $A = \{x_1, ..., x_{m+1}\}$ occurring on at least L bubbles of a P-expansion of π) guesses (1) an IDB predicate T of π, reachable from P (2) a path $T \xrightarrow{r_1} ... \xrightarrow{r_L} T'$ (of length L) and (3) a set of $m+1$ distinct terms. Also the algorithm verifies in NLINEARSPACE that m is the weak P-persistency number of program π.
(2) The problem is at least as hard as the (PSPACE-hard) problem NORMAL LINEAR DATALOG PROGRAM WEAK CHARACTERISTIC INTEGER. □

THEOREM 5.21. (1)The problem GENERAL LINEAR DATALOG PROGRAM WEAK CHARACTERISTIC INTEGER is PSPACE-complete, even for fixed L, and fixed weak persistency number, as long as it is at least 2.
(2) The problem GENERAL NON-LINEAR DATALOG PROGRAM WEAK CHARACTERISTIC INTEGER is APSPACE-complete, even for fixed L, and fixed weak persistency number, as long as it is at least 2.

PROOF 5.22. The proof for the hardness proof of (1) is based on the reduction described in the proof of Theorem 5.17, while the proof for the hardness proof of (2) is based on the reduction described in the proof of theorem 5.13; in both cases, we must add $L+1$ dummy rules as we did in the proof of theorem 5.19(1). The membership part of (1) is shown as in the proof of theorem 5.19(1). The membership part of (2) is shown similarly, using the proof of theorem 5.11 to compute the weak-P-persistency number of the program in ALINEARSPACE. □

6. DISCUSSION AND OPEN PROBLEMS

Persistent sets come up in the well-known "Magic Sets" transformation. In this context it seems relevant to maximize persistent variables before applying Magic Sets transformation: if the number of persistent variables is maximal then the subsequent application of the Magic Sets transformation gives more efficient programs.

The Magic Sets transformation is used to optimize the evaluation of Datalog programs in the particular case where variables of the goal predicate are bound to constants. Consider again the transitive closure query program
$P(x, y) \leftarrow E(x, z), P(z, y)$
$P(x, y) \leftarrow E(x, y)$

and consider as input database D the directed graph described by the facts $E(a,b)$, $E(b,c)$, $E(c,d)$, $E(a,e)$, $E(f,a)$; we can ask for several kind of queries, by binding or not certain variables: for instance the query $P(x,y)$ asks for all paths in D, while the query $P(a,y)$ asks for all paths in D starting from a and the query $P(x,d)$ asks for all paths in D ending up to d.

The query $P(a,y)$ is of type bf which means that the first argument is $b$ound and the second is $f$ree, while the query $P(x,d)$ is of (different) type fb which means that the second argument is $b$ound and the first is $f$ree. For answering such queries, where some variables in the goal predicate are bound to constants, not all rule instantiations (produced during bottom-up evaluation of the program) are necessary but only those instantiations "relevant" to the query i.e. those corresponding to a top-down processing of the rules starting from the goal; during that top-down processing of rules the initial bindings of the goal propagate from rule heads towards rule bodies, generating new bindings: the Magic Sets transformation uses the type of the query (i.e. the type of bindings in the goal predicate) for producing a new program, which mimics the propagation of bindings and which is more efficient because the rule instantiations it produces during bottom-up evaluation are those which are relevant to the query.

Persistent variables have an important impact on the Magic Sets transformation: indeed the program produced by the Magic Sets transformation is much simpler when the bound variables are the persistent ones; we explain the reason on the examples below.

First suppose that the bound variable is not a persistent one: consider for instance the query $P(a,y)$ where the non persistent variable x was bound to the value a; that binding is denoted by the fact $magic^{bf}(a)$ over a new predicate symbol $magic^{bf}$. When evaluating top-down the rule $P(a,y) \leftarrow E(a,z), P(z,y)$ the fact $E(a,b)$ gives the value b for the first argument z of $P(z,y)$: that binding is denoted by the fact $magic^{bf}(b)$; this propagation of bindings is expressed by the new rule $magic^{bf}(z) \leftarrow E(x,z), magic^{bf}(x)$, more precisely it is expressed by the rule instantiation $magic^{bf}(b) \leftarrow E(a,b), magic^{bf}(a)$ which produces $magic^{bf}(b)$ from $magic^{bf}(a)$. The propagation of bindings continues in a top-down way as long as the recursive rule of the initial program is used; each time, the first argument z of $P(z,y)$ will satisfy $magic^{bf}(z)$. The resulting program is the following, where the first rule just expresses the query $P(a,y)$:
$q^f(y) \leftarrow P^{bf}(a,y)$:
$magic^{bf}(a) \leftarrow$
$magic^{bf}(z) \leftarrow E(x,z), magic^{bf}(x)$
$P^{bf}(x,y) \leftarrow E(x,z), P^{bf}(z,y), magic^{bf}(x)$
$P^{bf}(x,y) \leftarrow E(x,y), magic^{bf}(x)$

Suppose now that the bound variable is the persistent variable y and consider for instance the query $P(x,d)$ where y is bound to the value d; that binding is denoted by the fact $magic^{fb}(d)$ over a new predicate symbol $magic^{fb}$. When evaluating top-down the rule $P(x,d) \leftarrow E(x,z), P(z,d)$ the second argument y of $P(z,y)$ immediately gets bound to the same value d because y persists from the head: that binding is denoted by the same fact $magic^{fb}(d)$; but y also persists during any number of unfoldings of the recursive rule i.e. y is a persistent variable and thus

y will always receive the same (initial) value d: the propagation of bindings is degenerated here and the Magic Sets transformation would produce a degenerate rule $magic^{fb}(y) \leftarrow magic^{fb}(y)$ with unique instantiation $magic^{fb}(d) \leftarrow magic^{fb}(d)$; after simplifications, the resulting program is the following, where the first rule just expresses the query $P(x, d)$:
$q^f(x) \leftarrow P^{fb}(x, d)$
$P^{fb}(x, d) \leftarrow E(x, d)$

That new program (produced for the query $P(x, d)$) is simpler than the one produced from the query $P(a, y)$: the reason is that the propagation of bindings on a persistent variable degenerates to the "propagation" of the same value, therefore there is no need to create a magic rule for describing it; instead of that, the specific value, d here, can be directly incorporated in the rules of the initial program.

The persistency number seems to be one reason contributing to high number of variables and a fortiori to a high space complexity. On the other hand, persistency number doesn't seem to be a refined enough notion to study computational complexity questions. It will be interesting to see whether combining the notion of persistent set with the more delicate notion of hypertree decomposition [Gottlob et al. 1999; 2001] might give some new perspectives on query evaluation.

Another point which is worth mentioning is the unbounded arities problem that comes up while proving the undecidability of the persistency number–modulo equivalence (in proof 4.5, the maximum arity of program $\pi_{G,m}$ depends on the persistency number). It would be interesting to find a proof using only bounded arities programs, but the existing literature on bounded versus unbounded arities has shown that such proofs are usually much more involved.

A different line of future research concerns inapproximability questions, and more precisely the following open problems:
(i) Is there some efficient transformation (in the spirit of Lemma 4.11) proving that for any $M > 0$, it is hard to distinguish between persistency number at most m and at least $m + M$.
(ii) Is there some efficient transformation proving that for any $L > 0$, it is hard to distinguish between weak characteristic integer at most l and at least $l + L$.

Finally, classes of Datalog programs with restricted persistency number should be investigated with respect to expressibility, decidability and complexity questions. A self-evident open problem is to extend the results of [Cosmadakis et al. 1988] on the decidability of boundedness for monadic Datalog programs, to classes of Datalog programs of persistency number 0.

ACKNOWLEDGMENTS

We would like to thank the anonymous referees for helpful comments and suggestions.

REFERENCES

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases.* Addison-Wesley.

Afrati, F. N. 1997. Bounded arity Datalog(≠) queries on graphs. *J. Comput. Syst. Sci. 55,* 2, 210–228.

Afrati, F. N. and Cosmadakis, S. S. 1989. Expressiveness of restricted recursive queries (ex-

tended abstract). In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, 15-17 May 1989, Seattle, Washington, USA*. ACM, 113–126.

Afrati, F. N., Cosmadakis, S. S., and Foustoucos, E. 2005. Datalog programs and their persistency numbers. *ACM Trans. Comput. Log. 6,* 3, 481–518.

Chandra, A. K., Kozen, D., and Stockmeyer, L. J. 1981. Alternation. *J. ACM 28,* 1, 114–133.

Cosmadakis, S. S., Gaifman, H., Kanellakis, P. C., and Vardi, M. Y. 1988. Decidable optimization problems for database logic programs (preliminary report). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, 2-4 May 1988, Chicago, Illinois, USA*. ACM, 477–490.

Cosmadakis, S. S. 1989. On the first-order expressibility of recursive queries. In *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 29-31, 1989, Philadelphia, Pennsylvania*. ACM, 311–323.

Courcelle, B. 1990. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*. 193–242.

Diestel. 2006. *Graph Theory*. Springer.

Gaifman, H., Mairson, H. G., Sagiv, Y., and Vardi, M. Y. 1987. Undecidable optimization problems for database logic programs. In *Proceedings, Symposium on Logic in Computer Science, 22-25 June 1987, Ithaca, New York, USA*. IEEE Computer Society, 106–115.

Gaifman, H., Mairson, H. G., Sagiv, Y., and Vardi, M. Y. 1993. Undecidable optimization problems for database logic programs. *J. ACM 40,* 3, 683–713.

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

Gottlob, G., Leone, N., and Scarcello, F. 1999. Hypertree decompositions and tractable queries. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*. ACM, 21–32.

Gottlob, G., Leone, N., and Scarcello, F. 2001. Hypertree decompositions: A survey. In *MFCS*, J. Sgall, A. Pultr, and P. Kolman, Eds. Lecture Notes in Computer Science, vol. 2136. Springer, 37–57.

Halin, R. 1976. *S*-functions for graphs. *J.of Geometry 8,* 1-2, 171–186.

Hillebrand, G. G., Kanellakis, P. C., Mairson, H. G., and Vardi, M. Y. 1991. Tools for datalog boundedness. In *Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 29-31, 1991, Denver, Colorado*. ACM, 1–12.

Hillebrand, G. G., Kanellakis, P. C., Mairson, H. G., and Vardi, M. Y. 1995. Undecidable boundedness problems for Datalog programs. *J. of Logic Programming 25,* 2, 163–190.

Hopcroft, J. E. and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Kolaitis, P. G. and Vardi, M. Y. 1998. Conjunctive-query containment and constraint satisfaction. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*. ACM, 205–213.

Marcinkowski, J. 1999. Achilles, turtle, and undecidable boundedness problems for small datalog programs. *SIAM J. Comput. 29,* 1, 231–257.

Ullman, J. D. 1988. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press.

Vardi, M. Y. 1988. Decidability and undecidability results for boundedness of linear recursive queries. In *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 21-23, 1988, Austin, Texas*. ACM, 341–351.

Received December 2006; revised November 2007; accepted June 2008