Theoretical Informatics and Applications Will be set by the publisher
Informatique Théorique et Applications

INF-DATALOG, MODAL LOGIC AND COMPLEXITIES.*

Eugénie Foustoucos[1] and Irène Guessarian[2]

Abstract. Inf-Datalog extends the usual least fixpoint semantics of Datalog with greatest fixpoint semantics: we defined inf-Datalog and characterized the expressive power of various fragments of inf-Datalog in [GFAA03]. In the present paper, we study the complexity of query evaluation on finite models for (various fragments of) inf-Datalog. We deduce a unified and elementary proof that global model-checking (i.e. computing all nodes satisfying a formula in a given structure) has 1. quadratic data complexity in time and linear program complexity in space for CTL and alternation-free modal μ-calculus, and 2. linear-space (data and program) complexities, linear-time program complexity and polynomial-time data complexity for $L\mu_k$ (modal μ-calculus with fixed alternation-depth at most k).

1991 Mathematics Subject Classification. 68Q19, 03C13.

1. Introduction

The model-checking problem for a logic $\mathcal{A}$ consists in verifying whether a formula φ of $\mathcal{A}$ is satisfied in a given structure $\mathcal{K}$. In computer-aided verification, $\mathcal{A}$ is a temporal logic *i.e.* a modal logic used for the description of the temporal ordering of events and $\mathcal{K}$ is a (finite) Kripke structure *i.e.* a graph equipped with

---

Keywords and phrases: databases, model-checking, specification languages, performance evaluation

* *Work partly supported by the project "Mathematical Logic, Recursion Theory and Applications", co-funded by the European Social Fund and National Resources - (EPEAEK II) PYTHAGORAS II*

[1] MPLA, National and Capodistrian University of Athens, Department of Mathematics, Panepistimiopolis, 15784 Athens, Greece.e-mail: aflaw@otenet.gr

[2] LIAFA, UMR 7089, Université Paris 7, case 7014, 2 Place Jussieu, 75251 Paris Cedex 5, France.**Corresponding author: e-mail:** ig@liafa.jussieu.fr

a labelling function associating with each node s the finite set of propositional variables of $\mathcal{A}$ that are true at node s.

Our approach to temporal logic model-checking is based on the close relationship between model-checking and Datalog query evaluation: a Kripke structure $\mathcal{K}$ can be seen as a relational database and a formula φ can be thought of as a Datalog query $\mathcal{Q}$. In this context, the model-checking problem for φ in $\mathcal{K}$ corresponds to the evaluation of $\mathcal{Q}$ on input database $\mathcal{K}$. The advantages of Datalog are that it is a simple declarative query language with clear semantics and low complexity (i.e., fixed Datalog programs can be evaluated in polynomial time over the input databases). When translated into Datalog, we thus can expect modal logic (e.g. μ-calculus) sentences to be easy to understand and check.

In [GFAA03] we introduced the language inf-Datalog, which extends usual least fixpoint semantics of Datalog with greatest fixpoint semantics: greatest fixpoints are necessary for expressing properties such as fairness (something must happen infinitely often). We translated into Monadic inf-Datalog various temporal logics (CTL, ETL, alternation-free modal μ-calculus, and modal μ-calculus [E90], by increasing order of expressive power); conversely we characterized the fragments of Monadic inf-Datalog which can be translated into these logics [GFAA03].

One of the advantages of inf-Datalog consists in eliminating problems inherent to negations: programs are assumed to be in positive normal form (negations affect only the explicitly given predicates); by duality, negation over computed predicates is expressed via greatest fixed points. Although inf-Datalog is syntactically very close to Datalog, computing the semantics of an inf-Datalog program (and consequently query evaluation) becomes more difficult conceptually and computationally; it is thus worth studying.

In the present paper we define an operational semantics for inf-Datalog and we give upper bounds for evaluating inf-Datalog queries: we describe an algorithm evaluating inf-Datalog queries and analyze its complexity with respect to the size of the database (*data complexity*) and its complexity with respect to the size of the program (*program complexity*). The data complexity is polynomial-time and linear-space. Using our succinct translations in [GFAA03] between the temporal logic paradigm and the database paradigm, we deduce upper bounds for the complexity of the model-checking problem for the modal μ-calculus.

2. Definitions

The basic definitions about Datalog can be found in [AHV95], [GFAA03], [GGV02], and the basic definitions about the μ-calculus and modal logic can be found in [AN01], [E90], [E97], [K83]. We proceed directly with the definition of inf-Datalog.

Definition 2.1. An *inf-Datalog program* is a Datalog program where some IDB predicates (*i.e.* predicates occurring in the heads of the rules) are tagged with an overline indicating that they must be computed as greatest fixed points; untagged IDB predicates are computed as least fixed points as usual; in addition, for each set of mutually recursive IDB predicates including both tagged and untagged IDB predicates, the order of evaluation of the IDB predicates in the set is specified by the indexing of the IDB predicates.

The dependency graph of a program is a directed graph with nodes the set of IDB predicates of the program; there is an edge from predicate ψ to predicate φ (denoted by $\varphi \longleftarrow \psi$) if there is a rule with head an instance of φ and at least one occurrence of ψ in its body, and φ is said to **directly depend on** ψ; φ is said to **depend on** ψ if there is a path from ψ to φ in the dependency graph (denoted by $\varphi \Longleftarrow \psi$). See Figures 4, 5 and examples 3.5, 3.6 for examples of dependency graphs. Two predicates that belong to the same strongly connected component of the dependency graph are said to be **mutually recursive**.

An inf-Datalog program is said to be **monadic** if all the IDB predicates have arity at most one. An inf-Datalog program is said to be **stratified** if no tagged IDB predicate is mutually recursive with an untagged IDB predicate. This notion of stratification is the natural counterpart (with respect to greatest fixed points) of

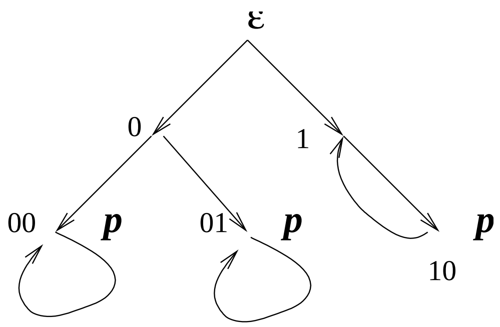

FIGURE 1. A structure of size 6

the well-known stratification with respect to negation; the denotational semantics of stratified inf-Datalog is the expected one; it is illustrated in Example 2.2.

Example 2.2. Consider as database the structure given in Figure 1, with two EDB predicates Suc_0 and Suc_1 denoting respectively the first successor and the second successor, and a unary EDB predicate p (which is meant to state some property of the nodes of the tree). Suc_0 is the relation

$$\{(\varepsilon, 0), (0, 00), (00, 00), (01, 01), (1, 10), (10, 1)\};$$

Suc_1 is the relation

$$\{(\varepsilon, 1), (0, 01), (00, 00), (01, 01), (1, 10), (10, 1)\};$$

p is assumed to hold for 00, 01 and 10, i.e. p is the relation $\{00, 01, 10\}$.

The program P below, has as IDB predicates $\overline{\theta}$ (computed as a greatest fixed point) and φ (computed as a least fixed point)

$$P\colon \begin{cases} \overline{\theta}(x) & \longleftarrow & p(x), Suc_0(x,y), Suc_1(x,z), \overline{\theta}(y), \overline{\theta}(z) & (1) \\ \varphi(x) & \longleftarrow & \overline{\theta}(x) & (2) \\ \varphi(x) & \longleftarrow & Suc_0(x,y), Suc_1(x,z), \varphi(y), \varphi(z) & (3) \end{cases}$$

The first stratum consists of rule 1 defining $\overline{\theta}$ (without initialization rule, this is possible because $\overline{\theta}$ is computed as a greatest fixed point); the second stratum defines φ as a least fixed point with rules 2 (initializing φ with the value computed for $\overline{\theta}$ in the first stratum) and 3. The IDB predicate $\overline{\theta}$ (resp. φ) in this program implements the modality $\mathbf{A}Gp$ (resp. $\mathbf{A}F\mathbf{A}Gp$) on the infinite full binary tree:

$\mathbf{A}Gp$ means that p is always true on all paths, and $\mathbf{A}F\mathbf{A}Gp$ means that, on every path we will eventually (after a finite number of steps) reach a state wherefrom p is always true on all paths. Gp is expressed by the CTL path formula $\bot\widetilde{\mathbf{U}}p$ and $\mathbf{A}F\mathbf{A}Gp$ is expressed by the CTL state formula $\mathbf{A}\big(\top\mathbf{U}\mathbf{A}(\bot\widetilde{\mathbf{U}}p)\big)$, where $\bot$ and $\top$ respectively represent *false* and *true*. The μ-calculus analog is the $L\mu_1$ expression $\mu\varphi.\Big(\nu\theta.(p\wedge\mathbf{A}\circ\,\theta)\bigvee\mathbf{A}\circ\,\varphi\Big)$. The semantics of P should (and does) yield $\{00,01\}$ as points where $\overline{\theta}$ holds and $\{0,00,01\}$ as points where φ holds.

The structures considered in temporal logics are usually infinite trees, while databases are always finite: how can we model the former with the latter? It should be noted that: (i) the infinite trees usually represent sequences of states occurring during the (infinite) execution of a (finite) program, hence they have a finite representation, and (ii) even with finite databases we can ensure that every node has a successor by adding a self-loop to every state without successor (as in states 00 and 01 of Example 2.2): we thus obtain the infinite sequences of states of temporal logic.

Remark 2.3. In Example 2.2 we assume that every node has outdegree at most 2; if the nodes have finite (but not known *a priori*) branching degree, then we slightly change our model, assuming two extensional binary predicates: $FirstSuc(x,y)$ ("y is the leftmost child of x"), and $NextSuc(x,y)$, ("y is the right sibling of x"). For instance, the formula $\varphi=\mathbf{A}\circ p$ (stating that p is true in all successors of a node) is expressed by the program:

$$P\colon\left\{\begin{array}{rcl} G_\varphi(x) & \longleftarrow & FirstSuc(x,y),T(y)\\ T(x) & \longleftarrow & p(x),NextSuc(x,y),T(y)\\ T(x) & \longleftarrow & p(x),\neg HasSuc(x)\\ HasSuc(x) & \longleftarrow & NextSuc(x,y)\end{array}\right.$$

Formula $\psi=\mathbf{E}\circ p$ (stating that p is true in some successor of a node) is expressed by program:

$$P\colon\left\{\begin{array}{rcl} G_\psi(x) & \longleftarrow & FirstSuc(x,y),T(y)\\ T(x) & \longleftarrow & p(x)\\ T(x) & \longleftarrow & NextSuc(x,y),T(y)\end{array}\right.$$

We now explain the more complex denotational semantics of non stratified inf-Datalog programs. Note first that we need not give any evaluation order within a set of IDB predicates that are all computed with the same fixed point (either least or greatest). We define the semantics of non-stratified programs by induction on the number k of alternations between mutually recursive least and greatest fixed points. If $k = 0$, either all IDBs are computed using least fixed points or all IDBs are computed using greatest fixed points, in the usual way. Assume the semantics of programs with at most k alternations of least and greatest fixed points is defined and let P be a program with $k+1$ such alternations. For instance, let $\Phi = \Phi^1 \cup \overline{\Phi^2} \cup \Phi^3 \cup \cdots \cup \overline{\Phi^{k+1}} \cup \Phi^{k+2}$ denote the set of IDBs of P, which are assumed to be mutually recursive; the order and type of evaluation are as follows: first all IDBs of Φ^1 are computed as least fixed points, then all IDBs of $\overline{\Phi^2}$ are computed as greatest fixed points, ..., and finally all IDBs of Φ^{k+2} are computed as least fixed points. Since the IDBs in $\Phi^1 \cup \overline{\Phi^2} \cup \Phi^3 \cup \cdots \cup \overline{\Phi^{k+1}}$ depend on the IDBs in Φ^{k+2}, the semantics of P is defined as follows: the IDBs in Φ^{k+2} are first considered as parameters, as in Gauss elimination method for solving systems of equations; let P_{k+1} be the program consisting only of those rules of P whose head is in $\Phi^1 \cup \overline{\Phi^2} \cup \Phi^3 \cup \cdots \cup \overline{\Phi^{k+1}}$ (and the IDBs in Φ^{k+2} are considered as EDBs). P_{k+1} has at most k alternations of least fixed points and greatest fixed points, hence can be solved formally by the induction hypothesis (with IDBs of Φ^{k+2} appearing in the solution). Then consider the set P'_{k+2} consisting of those rules of P whose head is in Φ^{k+2} and substitute in the corresponding rule bodies the solutions of P_{k+1} for the IDBs in $\Phi^1 \cup \overline{\Phi^2} \cup \Phi^3 \cup \cdots \cup \overline{\Phi^{k+1}}$: we obtain P'' where the only IDBs are those of Φ^{k+2}. Solve P'' and substitute the values obtained for the IDBs in Φ^{k+2} in the solutions of P_{k+1}. Iterate then these three steps (solving P_{k+1} and P'', substituting for the IDBs in Φ^{k+2} the values obtained when solving P''), until the least fixpoint of P'' is reached (*i.e.* the IDBs in Φ^{k+2} no longer change). Substitute finally this fixpoint for the IDBs of Φ^{k+2} occurring in P_{k+1}, thus obtaining the stabilized final values for the IDBs in $\Phi^1, \overline{\Phi^2}, \ldots, \overline{\Phi^{k+1}}$. This global process, just described, amounts to computing a $(k+2)$-ary fixpoint (i.e. until none of the IDBs in Φ changes anymore).

We will give an algorithm computing this semantics in Figure 6.

A related concept of semantics is defined for Horn clause programs with nested least and greatest fixpoints in [CANPW98]: their semantics is expressed directly in terms of the T_P operator. The authors of [CANPW98] consider as database the set of infinite ground terms allowing functions (i.e. the Herbrand universe), and the paper focuses on game-theoretic semantics.

3. Complexity of inf-Datalog

In the sequel we will count as one *basic time unit* the time needed to infer a single immediate consequence atom from a clause: *i.e.* assuming we have a clause $\varphi_i(x_1,\ldots,x_{n_i}) \longleftarrow \psi_1(x_{1,1},\ldots,x_{1,n_1}),\ldots,\psi_k(x_{k,1},\ldots,x_{k,n_k})$ and assuming that $\psi_1(x_{1,1},\ldots,x_{1,n_1}),\ldots,\psi_k(x_{k,1},\ldots,x_{k,n_k})$ hold ($\psi_1,\ldots,\psi_k$ can be IDBs or EDBs), we infer that $\varphi_i(x_1,\ldots,x_{n_i})$ holds in one basic time unit. Our time complexities will be counted relatively to that time unit.

Theorem 3.1. *Let P be a stratified program having I IDB symbols $\varphi_1,\ldots,\varphi_I$, and $\alpha = \max\{arity(\varphi_i), i = 1,\ldots,I\}$; let D be a relational database having n elements in its domain, then the set of* **all** *I queries defined by P and of the form (P,φ), where φ is an IDB of P, can be evaluated on D in time at most $C \times n^\alpha \times I$ and space at most $n^\alpha \times I$, assuming the time needed to evaluate $T_P(g_1,\ldots,g_I)$, g_i an arbitrary relation having the same arity as φ_i for $i = 1,\ldots,I$ is at most C (see Lemma 3.2).*

Proof. By induction on the number p of strata. We can assume without loss of generality that all IDBs in a stratum are of the same type, i.e. all untagged or all tagged. Assume P has a single stratum, and, e.g. all IDBs are untagged, hence computed as least fixed points. $T_P(g_1,\ldots,g_I)$ denotes the set of immediate consequences (obtained by a single application of a rule of P) when the meaning of φ_i is given by g_i, $i = 1,\ldots,I$. Let $\varphi_1,\ldots,\varphi_I$ be the IDBs, then the answer $f_1,\ldots,f_I$ to the set of queries $(P,\varphi_1),\ldots,(P,\varphi_I)$ defined by P is equal to $\sup_{i\in\mathbb{N}} T_P^i(\emptyset,\ldots,\emptyset)$ and, because D has n objects only, this least upper bound is obtained after at most n^α iterations, hence a time complexity at most $C \times n^\alpha \times I$. Similar proof if all IDBs are tagged (computed as greatest fixed points).

The case where P has p strata is similar: since the IDBs are computed in the order of the strata, assuming stratum j has I_j IDBs, the queries it defines will be computed in time at most $C \times n^\alpha \times I_j$, hence for the whole of P the complexity will be $C \times n^\alpha \times \sum_j I_j = C \times n^\alpha \times I$. The space complexity is clear too because we have at any time at most I IDBs true of at most n^α data objects. □

As pointed out by A. Arnold, Theorem 3.1 could also be obtained by first showing that inf-Datalog is a μ-calculus in the sense of [AN01], and then applying lemma 11.1.6 of [AN01] (extended to arbitrary structures).

Lemma 3.2. *Let D be a database with n elements in its domain and let P be an inf-Datalog program consisting of clauses of the form, $i = 1, \dots, I$:*

$$\begin{array}{rl}\varphi_i(x_1, \dots, x_{n_i}) \longleftarrow & \psi_1(x_1^1, \dots, x_{n_1}^1, y_1^1, \dots, y_{p_1}^1, c_1^1, \dots, c_{k_1}^1), \dots, \\ & \quad \psi_k(x_1^k, \dots, x_{n_k}^k, y_1^k, \dots, y_{p_k}^k, c_1^k, \dots, c_{k_k}^k)\end{array}$$

$x_1^j, \dots, x_{n_j}^j$ are variables among $x_1, \dots, x_{n_i}$ and $y_1^j, \dots, y_{p_j}^j$ are variables; $c_1^j, \dots, c_{k_j}^j$ are constants. Let $g_1, \dots, g_I$ be arbitrary relations having the same arities as $\varphi_1, \dots, \varphi_I$. The time needed to evaluate the i^{th} component of $T_P(g_1, \dots, g_I)$ on D^{n_i} is not greater than $C_i = N_i \times n^{m_i}$ where

$$\begin{array}{rl} N_i = & \text{the number of rules with head } \varphi_i \\ m_i = & \max\{\ |\{y_1^1, \dots, y_{p_1}^1, \dots, y_1^k, \dots, y_{p_k}^k\}| + |\{x_1^1, \dots, x_{n_1}^1, \dots, x_1^k, \dots, x_{n_k}^k\}| \ / \\ & \psi_1(x_1^1, ..., x_{n_1}^1, y_1^1, ..., y_{p_1}^1, c_1, ..., c_{k_1}^1), \dots, \psi_k(x_1^k, ..., x_{n_k}^k, y_1^k, ..., y_{p_k}^k, c_1^k, ..., c_{k_k}^k) \\ & \qquad \text{occurs in some rule with head } \varphi_i\} \end{array}$$

hence the time needed to evaluate any component ($i = 1, \dots, I$) of $T_P(g_1, \dots, g_I)$ is at most $C = \max_{i=1,\dots,I}(C_i)$.

Proof. Indeed evaluating the i^{th} component of $T_P(g_1, \dots, g_I)$ at a given point $(a_1, ..., a_{n_i}) \in D^{n_i}$ needs one basic time unit for each tuple of y_i^j's, such that

$$f_1(a_1^1, ..., a_{n_1}^1, y_1^1, ..., y_{p_1}^1, c_1^1, ..., c_{k_1}^1), \dots, f_k(a_1^k, ..., a_{n_k}^k, y_1^k, ..., y_{p_k}^k, c_1^k, ..., c_{k_k}^k)$$

holds for some rule with head φ_i: if ψ_i is φ_l then f_i will be g_l, otherwise f_i is the relation representing ψ_i, and if $x_i^j = x_l \in \{x_1, \dots, x_{n_i}\}$ then $a_i^j = a_l \in \{a_1, \dots, a_{n_i}\}$; therefore C_i basic time units at most are needed to compute the i^{th} component of $T_P(g_1, \dots, g_I)$ on D^{n_i}. □

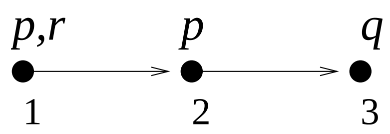

FIGURE 2. A structure with 3 elements

A trivial bound for C would be $N \times n^M$, where N is the number of rules and M is the maximum number of variables in rule bodies, but better bounds can be found in special cases of interest (e.g. programs corresponding to modal logic formulas). Even with this trivial bound, Theorem 3.1 implies that the data complexity of stratified inf-Datalog is PTime, hence adding greatest fixed points in a stratified way does not increase the evaluation complexity of Datalog (even though it increases its expressive power [GFAA03]).

Example 3.3. Consider the structure given in Figure 2, where $suc(1,2), suc(2,3), p(1),$ $p(2), q(3), r(1)$ hold and the Monadic Datalog program:

$$P\colon \begin{cases} \varphi(x) & \longleftarrow & q(x) \\ \varphi(x) & \longleftarrow & p(x), suc(x,y), \varphi(y) \\ \psi(x) & \longleftarrow & \varphi(x), r(x) \\ \psi(y) & \longleftarrow & \psi(x), suc(x,y) \end{cases}$$

Then, we need 6 steps to compute the queries defined by the program: $\varphi_0 = \emptyset, \varphi_1 = \{3\}, \varphi_2 = \{2,3\}, \varphi_3 = \{1,2,3\} = \varphi_4 = \varphi_5 = \varphi_6$. $\psi_0 = \psi_1 = \psi_2 = \psi_3 = \emptyset, \psi_4 = \{1\}, \psi_5 = \{1,2\}, \psi_6 = \{1,2,3\}$.

In the present example, $6 = n \times I$ is strictly less than the bound $C \times n \times I = 18 \times n \times I$ given in Theorem 3.1.

When all IDBs are evaluated as least fixed points (resp. greatest fixed points) the evaluation order of IDBs is irrelevant for the semantics: this is the reason why in plain Datalog, the evaluation order of IDBs is irrelevant; similarly in stratified inf-Datalog, the evaluation order of IDBs is irrelevant, because the semantics is computed stratum-wise, and in each stratum all IDBs have the same type, hence their evaluation order is irrelevant.

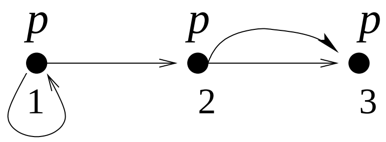

FIGURE 3. Another structure with 3 elements

We now turn to non-stratified Monadic inf-Datalog programs. Now, the situation changes and the evaluation order of mutually recursive IDBs becomes relevant. In the sequel, the order of evaluation of mutually recursive IDBs will be specified by their indexes, *i.e.* φ_i **will be computed before** φ_j **if and only if** $i < j$**.** As in μ-calculus, changing the evaluation order of IDBs can change the semantics of the program, see Example 3.5.

Definition 3.4. The syntactic alternation depth of program P is the largest k such that there exists in the dependency graph of P a cycle with alternations of k pairwise distinct tagged/untagged IDBs depending on each other, *e.g.* $\varphi_1 \Longrightarrow \overline{\varphi_2} \Longrightarrow \cdots \Longrightarrow \varphi_{k-1} \Longrightarrow \overline{\varphi_k} \Longrightarrow \varphi_1$ (all other combinations, i.e. φ_1 and/or φ_k tagged or untagged are allowed). It is 1 if there is no such cycle.

The syntactic alternation depth of programs we just defined corresponds to the syntactic alternation depth of μ-calculus formulas as defined in [BRA99], [S96], i.e. whenever program P is the translation of formula φ, their syntactic alternation depths are equal.

Example 3.5. Consider a 3-element structure having domain $\{1,2,3\}$, where $p(1), p(2), p(3)$, $Suc_1(1,1), Suc_0(1,2), Suc_0(2,3)$, and $Suc_1(2,3)$ hold (see Figure 3).

Let P_1 and P_2 be inf-Datalog programs, defined by:

$$P_1\colon \left\{ \begin{array}{lcll} X1(x) & \longleftarrow & p(x), Z3(x) & \\ X1(x) & \longleftarrow & p(x), Suc_i(x,y), X1(y) & \text{for } i = 0,1 \\ \overline{Y2}(x) & \longleftarrow & X1(x), p(x), Suc_i(x,y), \overline{Y2}(y) & \text{for } i = 0,1 \\ Z3(x) & \longleftarrow & \overline{Y2}(x) & \\ Z3(x) & \longleftarrow & Suc_0(x,y), Suc_1(x,z), Z3(y), Z3(z) & \end{array} \right.$$

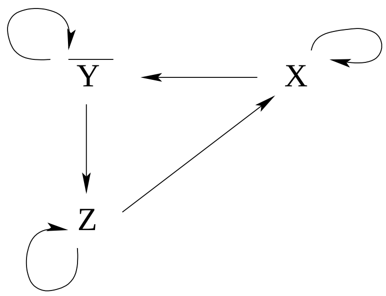

FIGURE 4. Dependency graph of P_1 and P_2

$$P_2\colon \begin{cases} Z1(x) & \longleftarrow & \overline{Y3}(x) \\ Z1(x) & \longleftarrow & Suc_0(x,y), Suc_1(x,z), Z1(y), Z1(z) \\ X2(x) & \longleftarrow & p(x), Z1(x) \\ X2(x) & \longleftarrow & p(x), Suc_i(x,y), X2(y) \qquad\qquad \text{for } i=0,1 \\ \overline{Y3}(x) & \longleftarrow & X2(x), p(x), Suc_i(x,y), \overline{Y3}(y) \qquad \text{for } i=0,1 \end{cases}$$

P_1 has 2 IDBs computed as least fixed points, $X1$ and $Z3$ and one IDB $\overline{Y2}$ computed as a greatest fixed point, the evaluation order is first $X1$ then $\overline{Y2}$ and last $Z3$; in the dependency graph we have the cycle $X_1 \longrightarrow \overline{Y_2} \longrightarrow Z_3 \longrightarrow X_1$, which can be reduced to $X_1 \longrightarrow \overline{Y_2} \Longrightarrow X_1$ and the syntactic alternation depth is $k=2$; P_2 has 2 IDBs computed as least fixed points, $X2$ and $Z1$ and one IDB $\overline{Y3}$ computed as a greatest fixed point, the evaluation order is first $Z1$ then $X2$ and last $\overline{Y3}$; in the dependency graph we have the cycle $Z_1 \longrightarrow X_2 \longrightarrow \overline{Y_3} \longrightarrow Z_1$, which can be reduced to $Z_1 \Longrightarrow \overline{Y_3} \longrightarrow Z_1$ and the syntactic alternation depth is $k=2$. If we forget the numbers indicating the evaluation order, both P_1 and P_2 have the same dependency graph, pictured in Figure 4. On the structure of Figure 3, P_1 computes $f_1 = f_2 = f_3 = \emptyset$, while P_2 computes $f_1 = f_2 = f_3 = \{1\}$.

Example 3.6. The μ-calculus sentences $\varphi \equiv \mu X.\nu Y.(X \vee Y \vee \mu Z.\nu W.(X \vee Z \vee (p \wedge W)))$ and $\psi \equiv \mu Y.\nu X.(X \vee Y \vee \mu Z.\nu W.(X \vee Z \vee (p \wedge W)))$ are respectively translated into inf-Datalog programs P_φ and P_ψ:

$$P_\varphi \begin{cases} \overline{W1}(x) & \longleftarrow & p(x), \overline{W1}(x) & \qquad \overline{Y3}(x) & \longleftarrow & Z2(x) \\ \overline{W1}(x) & \longleftarrow & X4(x) & \qquad \overline{Y3}(x) & \longleftarrow & \overline{Y3}(x) \\ \overline{W1}(x) & \longleftarrow & Z2(x) & \qquad \overline{Y3}(x) & \longleftarrow & X4(x) \\ Z2(x) & \longleftarrow & \overline{W1}(x) & \qquad X4(x) & \longleftarrow & \overline{Y3}(x) \end{cases}$$

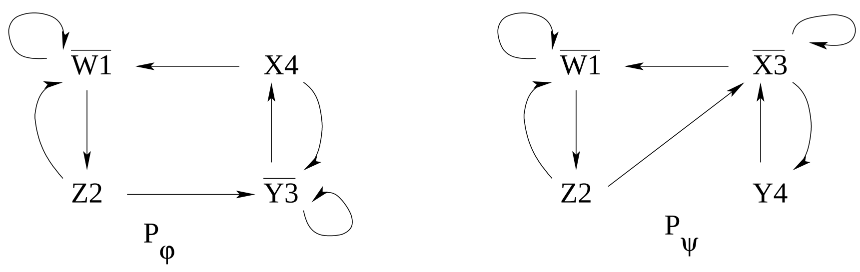

FIGURE 5. Dependency graphs of P_φ and P_ψ

$$P_\psi \left\{ \begin{array}{lllllll} \overline{W1}(x) & \longleftarrow & p(x), \overline{W1}(x) & \quad & \overline{X3}(x) & \longleftarrow & Z2(x) \\ \overline{W1}(x) & \longleftarrow & \overline{X3}(x) & & \overline{X3}(x) & \longleftarrow & Y4(x) \\ \overline{W1}(x) & \longleftarrow & Z2(x) & & \overline{X3}(x) & \longleftarrow & \overline{X3}(x) \\ Z2(x) & \longleftarrow & \overline{W1}(x) & & Y4(x) & \longleftarrow & \overline{X3}(x) \end{array} \right.$$

The translation is as in Corollary 4.2 and satifies moreover: (i) greatest (least) fixed points correspond to (un)tagged IDBs, (ii) IDBs are endowed with an index giving their evaluation order with the IDB corresponding to innermost (outermost) fixed points having lowest (highest) index, and (iii) subformulas of the form $\mu X.\nu Y.\psi$ give rules of the form $X_{i+1}(x) \longleftarrow \overline{Y_i}(x)$, and similarly for all other possible combinations of μ and ν. Both P_φ and P_ψ have syntactic alternation depth 4, and their dependency graphs are pictured in Figure 5 below; notice that arrows in the dependency graph go in the same direction as the corresponding arrows in the program.

Theorem 3.7. *Let P be a program with I recursive IDBs and syntactic alternation depth k fixed. Let α be the maximum arity of the IDBs φ_i, $i = 1, \dots, I$.*

Let D be a relational database having n elements. Then the set of **all queries** *of the form (P, φ), where φ is an IDB of P, can be computed on D in time $O\big(C \times n^{\alpha k} \times I\big)$ and space $O(n^\alpha \times I)$, where C is given in Lemma 3.2.*

Proof. The proof proceeds in three cases. To simplify we give the proof in the monadic case, i.e. when $\alpha = 1$.

Case 1. We will first study the case when $I = k$, therefore k even, φ_1 computed as a least fixed point (the algorithm for $\overline{\varphi_1}$ computed as a greatest fixed point is similar):

hence, there exist mutually recursive IDBs $\varphi_1, \overline{\varphi_2}, \ldots, \varphi_{k-1}, \overline{\varphi_k}$, computed in the order: first φ_1, then $\overline{\varphi_2}$, $\ldots$, and last $\overline{\varphi_k}$, with a cycle $\overline{\varphi_k} \Longrightarrow \varphi_1 \longrightarrow \overline{\varphi_2} \longrightarrow \cdots \longrightarrow \varphi_{k-1} \longrightarrow \overline{\varphi_k}$ in the dependency graph of P. (Both programs P_φ and P_ψ of Example 3.6 belong to case 1.) Let P'_i for $i > 1$ odd (resp. i even) be the set of rules of P with head φ_i (resp. $\overline{\varphi_i}$), and let P_1 be the set of rules with head φ_1. Then $P = P_k = P_1 \cup (\cup_{i=2}^{k} P'_i)$ has the following form:

$$P_k \left\{ \begin{array}{l} P'_k \left\{ \begin{array}{lll} \overline{\varphi_k}(x) & \longleftarrow & \cdots \\ & \cdots & \\ \overline{\varphi_k}(x) & \longleftarrow & \cdots \end{array} \right. \\ \\ P_{k-1} \left\{ \begin{array}{l} P'_{k-1} \left\{ \begin{array}{lll} \varphi_{k-1}(x) & \longleftarrow & \cdots \\ & \cdots & \\ \varphi_{k-1}(x) & \longleftarrow & \cdots \end{array} \right. \\ \\ \qquad \cdots \\ \\ \qquad P_2 \left\{ \begin{array}{l} P'_2 \left\{ \begin{array}{lll} \overline{\varphi_2}(x) & \longleftarrow & \cdots \\ & \cdots & \\ \overline{\varphi_2}(x) & \longleftarrow & \cdots \end{array} \right. \\ \\ P_1 \left\{ \begin{array}{lll} \varphi_1(x) & \longleftarrow & \cdots \\ & \cdots & \\ \varphi_1(x) & \longleftarrow & \cdots \end{array} \right. \end{array} \right. \end{array} \right. \end{array} \right.$$

The idea of the algorithm is similar to an algorithm given in [AN01] for evaluating boolean μ-calculus formulas and proceeds as follows. Let $f_1, \ldots, f_k$ be the answers on D to the queries defined by $\varphi_1, \ldots, \overline{\varphi_k}$. In order to compute f_k we must compute $\inf_i T^i_{P'_k}(\top)$, where $\top$ is true of every element in the data domain, and f_k will be reached after at most n steps (because the domain has n elements). However, since $\overline{\varphi_k}$ depends on φ_{k-1}, we must prealably compute $f_{k-1}[\top/\overline{\varphi_k}]$, which denotes f_{k-1} in which $\top$ has been substituted for the parameter $\overline{\varphi_k}$: this implies computing $\sup_i T^i_{P'_{k-1}}[\top/\overline{\varphi_k}](\emptyset)$, which is again reached after at most n steps, etc. The algorithm is described in Figure 6.

This algorithm consists of k nested loops `FOR` $j = 1$ `TO` $n+1$ `DO`. Notice that the innermost loop (j_1) is performed n times, whilst the $k-1$ outermost nested loops

```
ALGORITHM1
VAR j_1,...,j_k:  indices;
f_k := ⊤;
FOR j_k = 1 TO n+1 DO
    f_{k-1} := ∅;
    FOR j_{k-1} = 1 TO n+1 DO
        f_{k-2} := ⊤;
        ...
            f_2 := ⊤;
            FOR j_2 = 1 TO n+1 DO
                f_1 := ∅;
                FOR j_1 = 1 TO n DO
                    f_1 := T_{P_1} (f_1, f_2, ..., f_{k-1}, f_k);
                ENDFOR (j_1)
                IF j_2 ≤ n THEN f_2 := T_{P'_2} (f_1, f_2, ..., f_{k-1}, f_k);
            ENDFOR (j_2)
        ...
        IF j_{k-1} ≤ n THEN f_{k-1} := T_{P'_{k-1}}(f_1, f_2, ..., f_{k-1}, f_k);
    ENDFOR (j_{k-1})
    IF j_k ≤ n THEN f_k := T_{P'_k}(f_1, f_2, ..., f_{k-1}, f_k);
ENDFOR (j_k)
```

FIGURE 6. Algorithm1

are performed $n+1$ times each: for $j = j_2, \ldots, j_k$, each individual f_j is computed in at most $C \times n$ steps (because the time for computing each $T_{P'_j}(f_1, f_2, \ldots, f_{k-1}, f_k)$ is bounded by C). Then we have to add an $(n+1)$th round of iterations recursively reinitializing $f_k, \ldots, f_2$ by substituting the value just computed for f_j in $f_{j-1}, \ldots, f_1$, for $j = k, \ldots, 2$. At the end $f_1, \ldots, f_k$ will contain the answers to the queries defined by $\varphi_1, \ldots, \overline{\varphi_k}$. Example 3.10 shows that the final $(n+1)$th round of iterations can be necessary; hence the algorithm runs in time at most $C \times (n+1)^k \times I$ and its complexity is $O(C \times n^k \times I)$.

By Lemma 3.8, the algorithm is correct. The space complexity is at most $n \times I$ since we store the values of I IDBs, each of which can hold on at most n points.

Case 2. Assume now the set Φ of IDBs of P consists of I IDBs, $I > k$, which are all mutually recursive. Let $\Phi_1 \cup \overline{\Phi_2} \cup \Phi_3 \cup \cdots \cup \overline{\Phi_k}(\cup \Phi_{k+1})$ be a partition of Φ such that: 1) all the IDBs in Φ_i (resp. $\overline{\Phi_j}$) are untagged (resp. tagged), and 2) the order and type of evaluation are as follows: first all IDBs of Φ_1 are computed as least fixed points, then all IDBs of $\overline{\Phi_2}$ are computed as greatest fixed points, $\ldots$, and finally all IDBs of $\overline{\Phi_k}$ are computed as greatest fixed points (or if Φ_{k+1} is nonempty, all IDBs of Φ_{k+1} are computed as least fixed points). Both programs P_1 and P_2 of Example 3.5 belong to case 2, for P_1 the partition is $\Phi_1 \cup \overline{\Phi_2} \cup \Phi_3$ and for P_2 the partition is $\Phi_1 \cup \overline{\Phi_2}$. In the sequel we assume a partition of Φ of the form $\Phi_1 \cup \overline{\Phi_2} \cup \Phi_3 \cup \cdots \cup \overline{\Phi_k}$.

Assume that, for $i = 1, \ldots, k$, Φ_i has m_i IDBs $\varphi_{i,1}, \ldots, \varphi_{i,m_i}$ defined by program P'_i (tags omitted): $\Phi = \{\varphi_1, \ldots, \varphi_I\} = \cup_{i=1}^{i=k}\{\varphi_{i,1}, \ldots, \varphi_{i,m_i}\}$. Then (noting that $\{f_1, \ldots, f_I\} = \cup_{i=1}^{i=k}\{f_{i,1}, \ldots, f_{i,m_i}\}$) it suffices in algorithm1 to (i) replace each initialization (*e.g.* $f_i := \top$) with m_i initializations ($f_{i,j} := \top$, $j := 1, \ldots, m_i$), and (ii) substitute for each instruction: $f_i := T_{P'_i}(f_1, f_2, \ldots, f_i, \ldots, f_k)$ the set of m_i instructions:

$$\begin{array}{rcl} f_{i,1} & := & T_{P'_i,1}(f_1, f_2, \ldots, f_i, \ldots, f_I) \\ & \vdots & \\ f_{i,m_i} & := & T_{P'_i,m_i}(f_1, f_2, \ldots, f_i, \ldots, f_I) \end{array}$$

where $T_{P'_i,l}(f_1, f_2, \ldots, f_{i-1}, \ldots, f_I)$ denotes the set of immediate consequences which can be deduced using the rules of P'_i with head $\varphi_{i,l}$. At the end $f_1, \ldots, f_I$ will contain the answers to the queries defined by $\varphi_{1,1}, \ldots, \varphi_{1,m_1}, \ldots, \varphi_{k,1}, \ldots, \varphi_{k,m_k}$. Let t_k be the complexity of algorithm1 modified by (i)-(ii) for alternation depth k programs. Then $t_1 \leq C \times (m_1 \times n + m_1) = C \times (n+1) \times I$, and it is easy to check by induction that $t_k \leq C \times (n+1)^k \times I$: assuming $t_{k-1} \leq C \times (n+1)^{k-1} \times (I - m_k)$, we have $t_k \leq (t_{k-1} + C \times m_k) \times (n+1)$; by the induction hypothesis, $t_k \leq C \times (n+1)^k \times (I - m_k) + C \times m_k \times (n+1)$, and because $k \geq 2$, $m_k \times (n+1) < m_k \times (n+1)^k$, hence $t_k \leq C \times (n+1)^k \times I$.

The other cases: $\Phi = \Phi_1 \cup \overline{\Phi_2} \cup \Phi_3 \cup \cdots \cup \overline{\Phi_k} \cup \Phi_{k+1}$ and/or Φ_1 is a set $\overline{\Phi_1}$ of IDBs computed as greatest fixed points, are similar. The complexity of the algorithm is again $O(C \times n^k \times I)$.

Case 3. Last, in the most general case, the rules of P can be partitioned into $2n+1$ disjoint sets $\Sigma_0, \Pi_1, \Sigma_1, \ldots, \Pi_n, \Sigma_n$. Each $\Pi_i, i = 1, \ldots, n$, has I_i IDBs, all mutually recursive, and syntactic alternation depth $k_i \leq I_i$. Each $\Sigma_i, i = 0, \ldots, n$ is a (possibly empty) stratified program with J_i IDBs. The IDBs of Π_i can depend on the IDBs of Π_j and $\Sigma_j, j < i$ only; the IDBs of Σ_i can depend on the IDBs of Π_j and $\Sigma_j, j \leq i$ in a stratified way; no IDB of Π_i or Σ_i can depend on the IDBs of some Σ_j or $\Pi_j, j > i$. We evaluate first the queries defined by the IDBs of Σ_0 in time $O(C \times n \times J_0)$ (as in Theorem 3.1), then the queries defined by the I_1 mutually recursive IDBs of Π_1 in time $O(C \times n^{k_1} \times I_1)$ as in case 2, then the queries defined by the J_1 "stratified" IDBs of Σ_1 in time $O(C \times n \times J_1)$ as in Theorem 3.1, etc. Finally, the total time complexity is $O(C \times n^{\max k_i} \times (\sum_{i=1}^{n} I_i) + n \times (C \times \sum_{i=0}^{n} J_i)) = O(C \times n^k \times I)$, where k is the syntactic alternation depth of P (here $k = \max\{k_i | 1 \leq i \leq n\}$).

In all three cases, the space complexity of the algorithm is linear in both n and the number of IDBs which are being computed, hence the global space complexity is $O(n \times I)$. □

Lemma 3.8. *Let P be a program with syntactic alternation depth at most k, with IDBs $\varphi_1, \overline{\varphi_2}, \ldots, \overline{\varphi_k}$, and with parameters $g_1, \ldots, g_p$. Let D be a database with n elements. For any given values of the parameters $g_1, \ldots, g_p$, algorithm1 computes the answers $f_1, \ldots, f_k$ to the queries $\varphi_1, \overline{\varphi_2}, \ldots, \overline{\varphi_k}$ on D.*

Proof. By induction on k. We assume k even and the first IDB is a least fixed point, but the other cases (k odd, and/or $\overline{\varphi_1}$ is va greatest fixed point, and/or φ_k is a least fixed point) are similar.

Basis. For $k = 1$, the lemma is clear because there is no alternation.

Inductive step. Assume it holds for every $k' \leq k$ and prove it for $k+1$. Let $P_{k+1}(g_1,\ldots,g_p)$ be the program

$$P_{k+1}(g_1,\ldots,g_p) \left\{ \begin{array}{l} P'_{k+1} \left\{ \begin{array}{ll} \varphi_{k+1}(x) & \longleftarrow \cdots \\ & \cdots \\ \varphi_{k+1}(x) & \longleftarrow \cdots \end{array} \right. \\ \\ P_k(g_1,\ldots,g_p,\varphi_{k+1}) \end{array} \right.$$

Let $f_1(g_1,\ldots,g_p,\varphi_{k+1}),\ldots,f_k(g_1,\ldots,g_p,\varphi_{k+1})$ be the answers to the queries defined by IDBs $\varphi_1,...,\overline{\varphi_k}$ of $P_k(g_1,\ldots,g_p,\varphi_{k+1})$ (with parameters $g_1,\ldots,g_p$ *and* the current value of the relation φ_{k+1}, denoted also φ_{k+1}). For $i=1,\ldots,k$, we will denote each $f_i(g_1,\ldots,g_p,\varphi_{k+1})$ by $f_i(\varphi_{k+1})$. Let P'_{k+1} be the rules of $P_{k+1}(g_1,\ldots,g_p)$ with head φ_{k+1}. By the definition of nested fixed points, the answer f_{k+1} to the query defined by φ_{k+1} is the least upper bound of the sequence defined by $f^0_{k+1}=\emptyset$, $f^1_{k+1}=T_{P'_{k+1}}(f^0_{k+1})$, $\ldots$, $f^n_{k+1}=T_{P'_{k+1}}(f^{n-1}_{k+1})$. This sequence is computed in the outermost `FOR` loop of algorithm1 (for $k+1$): indeed, $f^1_{k+1} = T_{P'_{k+1}}(f^0_{k+1}) = T_{P'_{k+1}}(\emptyset) = T_{P'_{k+1}}[\emptyset/\varphi_{k+1}, f_k(\emptyset)/\overline{\varphi_k},\ldots,f_1(\emptyset)/\varphi_1]$ where $f_k(\emptyset),\ldots,f_1(\emptyset)$ are the answers to the queries defined by IDBs $\overline{\varphi_k},\ldots,\varphi_1$ of program $P_k(g_1,\ldots,g_p,\emptyset)$ with syntactic alternation depth $\leq k$. Similarly, for $j=1,\ldots,n-1$, each of $f_k(f^j_{k+1})$, $\ldots,f_1(f^j_{k+1})$ are the answers to the queries defined by IDBs $\overline{\varphi_k},\ldots,\varphi_1$ of program $P_k(g_1,\ldots,g_p,f^j_{k+1})$ and $f^{j+1}_{k+1}=T_{P'_{k+1}}(f^j_{k+1})=T_{P'_{k+1}}[f^j_{k+1}/\varphi_{k+1}, f_k(f^j_{k+1})/\overline{\varphi_k},...,$ $f_1(f^j_{k+1})/\varphi_1]$. As explained above, f^n_{k+1} is the answer f_{k+1} to the query φ_{k+1} defined by the program $P_{k+1}(g_1,\ldots,g_p)$; hence, the answers $f_k,\ldots,f_1$ to the queries $\overline{\varphi_k},\ldots,\varphi_1$ defined by the program $P_{k+1}(g_1,\ldots,g_p)$ are respectively $f_k[f^n_{k+1}/\varphi_{k+1}]$, $\ldots,f_1[f^n_{k+1}/\varphi_{k+1}]$ which are computed in the final round of iterations. $\square$

The next Corollary follows from Theorem 3.7.

Corollary 3.9. *The set of queries defined by inf-Datalog programs can be computed in time polynomial in the number of elements of the database, exponential in the number of variables and the syntactic alternation depth of the program, and linear in the number of IDBs. The space complexity is linear in I and polynomial in n (where n is the number of elements of the structure, and I is the number of IDBs).*

Example 3.10. Consider the same structure as in Figure 3, and the program P below (where $I = 2 = k$):

$$P: \begin{cases} \overline{\varphi^2}(x) & \longleftarrow & \theta^1(x), Suc_0(x,y), Suc_1(x,z), \overline{\varphi^2}(y), \overline{\varphi^2}(z) \\ \theta^1(x) & \longleftarrow & Suc_i(x,y), \theta^1(y) \qquad \text{for } i = 0,1 \\ \theta^1(x) & \longleftarrow & p(x), \overline{\varphi^2}(x) \end{cases}$$

Then algorithm1 will compute: 1. for $f_2 = \top$, $f_1 = \{1,2,3\}$, and $f_2 = \{1,2\}$; then, 2. for $f_2 = \{1,2\}$, $f_1 = \{1,2\}$, and $f_2 = \{1\}$; then, 3. for $f_2 = \{1\}$, $f_1 = \{1\}$, and $f_2 = \emptyset$; a last round will give 4. for $f_2 = \emptyset$, $f_1 = \emptyset$. P is the translation of the temporal logic formula: $\varphi = \mathbf{E}(F^\infty p \wedge \mathbf{A} \circ \ F^\infty p)$ expressing that there exists a path on which p holds infinitely often and moreover, on all successors of the first state of that path, again p holds infinitely often. This formula cannot hold on a finite structure without infinite paths as the structure in Figure 3.

In [AN01], [EL86] finer notions of alternation depth are defined: they correspond to counting only the alternations which affect the semantics because the innermost fixed point depends on the outermost fixed point; algorithmically, this leads to an improvement by computing beforehand and only once the fixed points associated with closed subformulas. Our algorithm1 can be improved to match the finer notion of [EL86], which we first recall.

Definition 3.11. Given a set P of propositional variables and a set Ξ of variables, the set M of modal μ-calculus formulas is inductively defined by:

$$M ::= \begin{array}{l} \{p | p \in P\} \cup \{\neg p | p \in P\} \cup \{X | X \in \Xi\} \\ \cup \{\varphi \vee \psi, \varphi \wedge \psi, \mathbf{A} \circ \varphi, \mathbf{E} \circ \varphi | \varphi, \psi \in M\} \cup \{\mu X.\varphi, \nu X.\varphi | X \in \Xi, \varphi \in M\} \end{array}$$

The semantic alternation depth $ad(\varphi)$ of formula φ is defined by: (i) $ad(\varphi) = 0$ if $\varphi \in P \cup \Xi$, and $ad(\neg p) = 0$ if $p \in P$, (ii) $ad(\mathbf{A} \circ \varphi) = ad(\mathbf{E} \circ \varphi) = ad(\varphi)$, (iii) $ad(\varphi \vee \psi) = ad(\varphi \wedge \psi) = \max(ad(\varphi), ad(\psi))$ and (iv) $ad(\mu X.\varphi) = \max(ad(\varphi), 1 + \max\{ad(\psi) | \psi = \nu Y.\phi$ is a subformula of φ and X occurs free of any μ or ν binding in $\psi\})$, and similarly $ad(\nu X.\varphi) = \max(ad(\varphi), 1 + \max\{ad(\psi) | \psi = \mu Y.\phi$ is a subformula of φ and X occurs free of any μ or ν binding in $\psi\})$.

We now define a notion of semantic alternation depth corresponding to the definition of [EL86]. We study here the case when the number of IDBs is equal to the syntactic alternation depth of the program. The other cases can be treated similarly but at the cost of heavier notations.

Definition 3.12. Let P be a program with syntactic alternation depth k, having mutually recursive IDBs $\varphi_1, \overline{\varphi_2}, \ldots, \overline{\varphi_k}$, with a cycle $\varphi_1 \longrightarrow \overline{\varphi_2} \longrightarrow \cdots \longrightarrow \varphi_{k-1} \longrightarrow \overline{\varphi_k} \Longrightarrow \varphi_1$ in the dependency graph of P. (All other combinations, i.e. φ_1 and/or φ_k tagged or untagged are allowed.) The semantic alternation depth of φ_i, denoted by $ad(\varphi_i)$, is defined as follows: $ad(\varphi_1) = 1$ and $ad(\varphi_i) = ad(\overline{\varphi_{i-1}})$ $+ \, e$ where

$$e = \begin{cases} 1, & \text{if in the dependency graph of } P, \text{ there is an edge } \varphi_i \longrightarrow \overline{\varphi_j} \\ & (\text{ or } \varphi_i \longrightarrow \varphi_j), j < i \\ 0, & \text{otherwise} \end{cases}$$

The semantic alternation depth of $\overline{\varphi_i}$ is defined similarly.

The semantic alternation depth of P, denoted by $ad(P)$ is equal to the semantic alternation depth of $\overline{\varphi_k}$ (resp. φ_k if $\overline{\varphi_1}$ is tagged).

Example 3.13. Let φ and ψ be as in Example 3.6. Formulas φ and ψ, programs P_φ and P_ψ all have the same syntactic alternation depth 4, but their semantic alternation depths differ. In P_φ, $ad(P_\varphi) = ad(X4) = 3$ and $ad(\overline{Y3}) = ad(Z2) = 2$, while in P_ψ, $ad(P_\psi) = ad(Y4) = 4$ and $ad(\overline{X3}) = 3$, $ad(Z2) = 2$. The (semantic) alternation depth of formula φ in the sense of [EL86] is also 3, whilst it is only 2 in the sense of [AN01]. The (semantic) alternation depth of formula ψ is 4 in the sense of both [AN01], [EL86].

Theorem 3.14. *Let P be a program with syntactic alternation depth k, IDBs $\varphi_1, \overline{\varphi_2}, \ldots, \overline{\varphi_k}$, and parameters $g_1, \ldots, g_p$. To simplify P is assumed to be monadic. Let D be a database with n elements. For any given values of the parameters $g_1, \ldots, g_p$, we can compute the answers $f_1, \ldots, f_k$ to the queries $\varphi_1, \ldots, \overline{\varphi_k}$ on D in space $O(n \times k)$ and time $O(C \times n^d \times k)$, where $d = ad(P)$ is the semantic alternation depth of P and C is given in Lemma 3.2.*

Proof. It is similar to Case 1 of Theorem 3.7, but uses a slight improvement of Algorithm1: some iterations computing the f_is need not be nested (we avoid recomputing f_is when their value does not change). By induction on j, we prove (the tags will eventually be omitted in the proof):

Fact: For $j \geq 1$, considering the current values of $\varphi_{j+1}, \ldots, \overline{\varphi_k}$ as parameters (and denoting them again by $\varphi_{j+1}, \ldots, \overline{\varphi_k}$), we compute the answers $f_1(g_1, \ldots, g_p, \varphi_{j+1}, \ldots, \overline{\varphi_k}), \ldots, f_j(g_1, \ldots, g_p, \varphi_{j+1}, \ldots, \overline{\varphi_k})$ to the queries $\varphi_1, \ldots, \varphi_j$ (defined by $P_j(g_1, \ldots, g_p, \varphi_{j+1}, \ldots, \overline{\varphi_k}))$ in time $O(C \times n^{ad(\varphi_j)} \times j)$.

Basis: If $j = 1$, $ad(\varphi_1) = 1$ and we have to compute $f_1(g_1, \ldots, g_p, \varphi_2, \ldots, \varphi_k)$ as a least upper bound, which can be done in at most $C \times n$ steps.

Induction: Assume the result holds for $j \geq 1$ and prove it for $j + 1$. We distinguish two cases:

case (i): in the dependency graph of P there is an edge $\varphi_{j+1} \longrightarrow \varphi_i$, for some $i < j + 1$; hence $ad(\varphi_{j+1}) = ad(\varphi_j) + 1$; moreover, as there is always a path $\varphi_i \Longrightarrow \varphi_j$ in the dependency graph (because all IDBs are mutually recursive), φ_j thus depends on φ_{j+1} and $f_j(g_1, \ldots, g_p, f^q_{j+1}, \varphi_{j+2}, \ldots, \varphi_k)$ is *a priori* different from $f_j(g_1, \ldots, g_p f^{q+1}_{j+1}, \varphi_{j+2}, \ldots, \varphi_k)$; hence the `FOR` j_{j+1} loop of algorithm1 must be nested over the loops computing $f_1, \ldots, f_j$; as by the induction hypothesis the latter are computed in time $O(C \times n^{ad(\varphi_j)} \times j)$, together with the final englobing loop `FOR` j_{j+1}, $f_1, \ldots, f_j, f_{j+1}$ will be computed in time $O(C \times n^{ad(\varphi_j)} \times j \times (n+1)) = O(C \times n^{ad(\varphi_j)+1} \times j) = O(C \times n^{ad(\varphi_{j+1})} \times (j + 1))$.

case (ii): there is no edge from φ_{j+1} to some φ_i, $i < j + 1$; then $ad(\varphi_{j+1}) = ad(\varphi_j)$ and none of the IDBs φ_i, $i \leq j$ depends directly on φ_{j+1}: thus, for every $i \leq j$, each $f_i(g_1, \ldots, g_p, f^q_{j+1}, \varphi_{j+2}, \ldots, \varphi_k)$ is *a priori* equal to $f_i(g_1, \ldots, g_p, f^{q+1}_{j+1}, \varphi_{j+2}, \ldots, \varphi_k)$, and the `FOR` j_{j+1} loop of algorithm1 does not need to be nested over the loops computing $f_1, \ldots, f_j$: it suffices to perform it after completion of these loops, in time $O(C \times n)$; as by the induction hypothesis the latter are computed in time $O(C \times n^{ad(\varphi_j)} \times j)$, the time for computing $f_1, \ldots, f_j, f_{j+1}$ will be $O(C \times n^{ad(\varphi_j)} \times j + n \times C) = O(C \times n^{ad(\varphi_j)} \times (j+1)) = O(C \times n^{ad(\varphi_{j+1})} \times (j+1))$.

Letting $j = k$ gives the time complexity stated in the theorem; the space complexity is clear. $\square$

From Lemma 3.2 and Theorem 3.14, we deduce

Corollary 3.15. *The set of queries defined by inf-Datalog programs can be computed in time polynomial in the number of elements of the database, exponential in the number of variables and the semantic alternation depth of the program, and linear in the number of IDBs.*

4. Monadic inf-datalog and Modal Logic

In the present section, all programs will be Monadic inf-Datalog programs.

Recall the *model-checking* problem: given a formula φ, a structure M and an element (node) s of M, to check whether φ holds at node s of structure M. We will solve the slightly more general problem, which we call **global model-checking**: given formula φ and structure M, to compute the set of **all** nodes s of M such that φ holds at node s.

As a consequence of Theorem 3.1 we get an upper bound for the complexity of model checking for CTL and various modal logics. We first state a lemma improving Lemma 3.2 for Modal inf-Datalog.

Lemma 4.1. *Let D be a database with n elements in its domain and let P be a Modal inf-Datalog program with IDBs $\varphi_1, \dots, \varphi_I$. Let $g_1, \dots, g_I$ be arbitrary relations having the same arities as $\varphi_1, \dots, \varphi_I$. The time needed to evaluate any component ($i = 1, \dots, I$) of $T_P(g_1, \dots, g_I)$ is $C = O(n)$.*

Proof. Recall [GFAA03] that a *Modal inf-Datalog program* P is an inf-Datalog program such that

- all predicates of P are **monadic** with the exception of Suc_0, Suc_1, and
- all rules of P have one of the following forms

(i) $\varphi(x) \longleftarrow un(x), Suc_i(x,y), \psi'(y),$ for $i = 0,1$ and $x \neq y$

$$
\text{(ii)} \begin{cases} \varphi(x) \longleftarrow un(x), Suc_0(x,y), Suc_1(x,z), \psi'(y), \psi'(z), \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{and } x,y,z \text{ pairwise distinct} \\ \varphi(x) \longleftarrow un(x), Suc_0(x,y), \neg 2Suc(x), \psi'(y), \qquad \text{and } x \neq y \\ 2Suc(x) \longleftarrow Suc_0(x,y), Suc_1(x,z) \end{cases}
$$

$$
\text{(iii)} \ \varphi(x) \longleftarrow un(x)
$$

where ψ' needs not be different from φ and where $un(x)$ is a sequence of monadic literals $\psi(x)$ where $\psi(x)$ is either an IDB atom with $\psi \neq \varphi$ or an atomic proposition $p(x)$ or its negation $\neg p(x)$.

For Modal inf-Datalog programs, we note that: (i) for any x in M there is at most one y such that $Suc_i(x,y)$ holds, (ii) each rule body can be true for at most one tuple (which we assume to be built-in and which we do not have to compute), and (ii) there are at most 3 rules for each non-terminal. Hence the time needed to evaluate one $\varphi(a)$ is one unit of time, and we can improve the bound C of Lemma 3.2: the time needed to evaluate one component of $T_P(g_1,\ldots,g_I)$ becomes $O(n)$. □

Corollary 4.2. *The global model-checking problem for CTL, ETL and alternation-free modal μ-calculus can be solved in time $O(n^2 \times |\varphi|)$ and space $O(n \times |\varphi|)$, where n is the number of elements in the domain of model M and $|\varphi|$ is the size of the formula.*

Proof. We can translate every CTL sentence φ into a stratified Modal inf-Datalog program P such that (1) the structure (M,s) is a model of φ (i.e. φ holds at node s of structure M) if and only if s is an answer to query $G_\varphi(x)$ defined by P on M, and (2) the number I of IDBs of P is less than the size of φ. We treat below a (non exhaustive but typical) set of formulas illustrating the basic steps of our translation.

- $\varphi \equiv p_1 \wedge p_2$ is translated into $G_\varphi(x) \longleftarrow p_1(x), p_2(x)$
- $\varphi \equiv p_1 \vee p_2$ becomes $G_\varphi(x) \longleftarrow p_1(x)$, $G_\varphi(x) \longleftarrow p_2(x)$
- $\varphi \equiv \mathbf{E} \circ \mathbf{p}$ becomes $G_\varphi(x) \longleftarrow Suc_i(x,y), p(y)$ for $i = 0, 1$
- $\varphi \equiv \mathbf{A} \circ \mathbf{p}$ becomes the 3-rule program:

$$\begin{cases} G_\varphi(x) & \longleftarrow & Suc_0(x,y), \neg 2Suc(x), p(y) \\ G_\varphi(x) & \longleftarrow & Suc_0(x,y), Suc_1(x,z), p(y), p(z) \\ 2Suc(x) & \longleftarrow & Suc_0(x,y), Suc_1(x,z) \end{cases}$$

- $\varphi \equiv \mathbf{E}(p_1 \mathbf{U} p_2)$ becomes the 3-rule program:
$$\begin{cases} G_\varphi(x) & \longleftarrow & p_2(x) \\ G_\varphi(x) & \longleftarrow & p_1(x), Suc_i(x,y), G_\varphi(y) \text{ for } i=0,1 \end{cases}$$
- $\varphi \equiv \mathbf{E}(p_1 \tilde{\mathbf{U}} p_2)$ becomes:
$$\begin{cases} \overline{G_\varphi}(x) & \longleftarrow & p_1(x), p_2(x) \\ \overline{G_\varphi}(x) & \longleftarrow & p_2(x), Suc_i(x,y), \overline{G_\varphi}(y) \text{ for } i=0,1 \end{cases}$$

ETL and alternation-free μ-calculus sentences can be similarly translated [GFAA03] into stratified Modal inf-Datalog, hence the result by Lemma 4.1.

The restriction that every node has outdegree at most 2 is inessential, see Remark 2.3. □

The best known bound for model-checking is linear in the size $|M| = n + T$ of the model (where T is the number of tuples in the database M); see [GGV02], and also [AC88], [CES86], [CS93].

Similarly, from Theorem 3.14 we can deduce

Corollary 4.3. *The global model-checking problem for the modal μ-calculus can be solved in time polynomial in the number of elements in the domain of the model and exponential in the semantic alternation depth of the formula. The space complexity is linear in $n \times |\varphi|$ (where n is the number of elements in the domain of the model and $|\varphi|$ the size of the formula).*

Proof. We detail the proof when $\varphi = \theta_k X_k.\varphi_k$, is a modal μ-calculus sentence of syntactic alternation depth k containing exactly k fixed points, i.e. for $i = 2, \ldots, k+1$ (where $\varphi_{k+1} = \varphi_1$): $\varphi_i = g_i(X_1, \ldots, \theta_{i-1} X_{i-1}.\varphi_{i-1}, \ldots, X_k)$, where g_i is a modal μ-calculus formula and $\{\theta_i, \theta_{i-1}\} = \{\mu, \nu\}$, *i.e.* $\theta_{i-1} \neq \theta_i$.

As in Corollary 4.2, we translate φ into a Monadic inf-Datalog program P_φ (see [GFAA03]) such that (i) C is in $O(n)$, (ii) the semantic alternation depth $ad(P_\varphi)$ of P_φ is equal to $ad(\varphi)$ as defined in [EL86] (see *e.g.* Examples 3.6 and 3.13), and (iii) the number of IDBs of P_φ is less than the size of φ. Because of the form of φ and because φ has syntactic alternation depth k, there is a cycle

$\overline{X_k} \Longrightarrow X_1 \longrightarrow \overline{X_2} \longrightarrow \cdots \longrightarrow X_{k-1} \longrightarrow \overline{X_k}$ in the dependency graph of P_φ, hence the syntactic alternation depth of P_φ is k. It is easy to check by induction on k that the semantic alternation depth $ad(P_\varphi)$ of P_φ is equal to the semantic alternation depth $ad(\varphi)$ of φ: we only have to observe that (i) X_i occurs free (of any μ or ν binding) in φ_j, for $j < i$, iff X_i occurs free in a φ_s, $s \leq j$, and (ii) s is the least index such that X_i occurs free in φ_s, $s \leq j$, iff there is in the dependency graph of P_φ an edge $X_i \longrightarrow X_s$ (or $X_i \longrightarrow \overline{X_s}$). Then, because C is in $O(n)$, Theorem 3.14 implies that model-checking for φ is in time $O((n)^{1+ad(\varphi)} \times |\varphi|)$ and space $O(n \times |\varphi|)$, hence the corollary.

For the general case, (i) we first generalize the notion of semantic alternation depth to any form of Monadic inf-Datalog program P, and extend Theorem 3.14 to programs belonging to cases 2 and 3 of Theorem 3.7 (the time and space complexities respectively become $O(n^{ad(P)+1} \times I)$ and $O(n \times I)$), and (ii) we then check (as above) that the translation from an arbitrary μ-calculus sentence φ to P_φ still preserves the semantic alternation depth. Whence the corollary. $\square$

5. Discussion and Conclusion

We gave a polynomial-time algorithm computing the set of *all* answers to the queries defined by an inf-Datalog program. For Modal inf-Datalog programs, the worstcase time complexity of this algorithm is $O(n^{k+1} \times I)$ where n is the number of elements in the domain of the database, I the number of IDBs and k the semantic alternation depth. Because Modal inf-Datalog subsumes the modal μ-calculus, we deduced new proofs of the complexity of model-checking μ-calculus formulas. Our upper bounds are given by polynomials whose degree is higher by one than the degree of the improved upper bound for model checking given in [E97]: this is due to the fact that we not only check whether a given node s of structure M satisfies formula φ (as in model checking), but **we compute the set of all nodes** in M that satisfy φ. Note also that our bounds are given with respect to the number n of elements of M, and not as usual, with respect to the size $|M| = n + T$ (T is the number of tuples) of M.

Related ideas can be found in the following papers:

1) Inf-Datalog is equivalent to a fragment of the mu-calculus of [P76], which has only least fixed points, but allows for even numbers of negations and non-Horn formulas; this calculus has been implemented [GV04], [V03] using BDDs.

2) [M95] translates model-checking a modal μ-calculus sentence into solving a boolean equation system, which is then solved by Gauss elimination method (similar to our denotational semantics given in Section 2); the complexity of solving the boolean equations is not studied, but an optimisation consisting of solving only those equations necessary to evaluate the variable one wants to compute is proposed. Mader also extended her work to infinite equation systems (corresponding to infinite models).

3) [S96] also reduces model-checking a modal μ-calculus sentence f to solving fixed point equations; he proposes to first unfold syntactically the fixed point equations in order to remove some iterations, and this enables him to solve them in time $O((|M| \times |f|)^{1+\lceil k/2 \rceil})$.

4) [AN01] defines an algorithm close to our algorithm1 of Figure 6 for computing vectorial boolean fixed point formulas.

Our programs are in positive normal form, meaning negations can affect only the explicitly given predicates and not the computed IDBs; the subset of μ-calculus formulas thus captured is a strict subset of the whole μ-calculus; however, all μ-calculus *sentences* can be put in positive normal form, [AN01] page 146; because model-checking concerns sentences only, we can assume positive normal forms without loss of generality for our purpose, and we gain the fact that problems caused by negations (even number of negations, complex semantics, etc.) vanish.

The only counterpart is that the negations are incorporated in the greatest fixpoints, hence we have to be careful and give explicitly (as we did) the order of evaluation of alternating fixed points. The approach allowing explicit negations is taken in the seminal papers [K83], [E97], [EL86]; in more recent work [AN01], [S96], [M95] only positive normal forms are allowed.

For model-checking formulas of CTL and the alternation-free μ-calculus (which are equivalent to fragments of stratified Modal inf-Datalog), [GGV02] also gives a linear-time algorithm, through a translation into Datalog LITE (an extension of Datalog using universal quantifications in rule bodies). However, Datalog LITE is essentially alternation-free and does not capture CTL*, nor LTL, nor the modal μ-calculus.

$L\mu_k$ is the set of modal μ-calculus formulas of syntactic alternation depth k; it is equivalent to a fragment of Modal inf-Datalog. For model-checking a formula f of $L\mu_k$: (i) [EL86] states a time complexity in $O\big((|M| \times |f|)^{k+1}\big)$, (ii) [E97] describes a semi-naive algorithm avoiding some redundant computations and running in time $O\big((|M| \times |f|)^{k}\big)$. We obtain for *global* model-checking a running time $O\big(n^{k+1} \times |f|\big)$, where k is the semantic alternation depth of f, by combining Theorem 3.14 and our translation [GFAA03] from the modal μ-calculus into Modal inf-Datalog. We believe that this bound could be slightly improved: indeed [BCJLM97] gives an algorithm for model-checking formulas of $L\mu_k$ running in time $O\big((|M| \times |f|)^{2+k/2}\big)$; however the space complexity of the improved algorithm in [BCJLM97] is exponential whilst the space complexity of the (semi-)naive algorithms is polynomial [E97].

Our algorithms are stated directly in terms of Inf-Datalog programs and database instances, which makes them transparent and potentially useful for Inf-Datalog compilers. This however comes at the expense of optimality. By reducing the problem to parity games, [J00] gives an algorithm running in time $O\big((|M| \times |f|)^{2+k/2}\big)$ and in polynomial space. This is currently the best time complexity for model-checking formulas of $L\mu_k$: indeed, for large ks ($k = \Omega(n^{1/2+\varepsilon})$) a better algorithm for parity games has been given in [JPZ06]; however the algorithm in [J00] outperforms the one in [JPZ06] in our case, because k will be small (most natural formulas have a syntactic alternation depth of at most 2).

We conjecture that inf-Datalog could be extended to capture the whole μ-calculus of [P76], while retaining a polynomial time data complexity.

Acknowledgments: We thank the referee for helpful comments. The second author is very grateful to Damian Niwiński for enlightening discussions and to André Arnold for his reading and comments.

References

[AHV95] S. Abiteboul, R. Hull, V. Vianu, *Foundations of databases*, Addison-Wesley, 1995.

[AC88] A. Arnold, P. Crubillé, *A Linear Algorithm to Solve Fixed-Point Equations on Transition Systems*, **IPL**, 29 (2), 1988, 57-66.

[AN01] A. Arnold, D. Niwiński, *Rudiments of μ-calculus*, Elsevier Science, Studies in Logic and the Foundations of Mathematics, 146, North-Holland, Amsterdam, 2001.

[BRA99] J. Bradfield, *Fixpoint alternation: Arithmetic, transition systems, and the binary tree*, **RAIRO**, Theoretical Informatics and Applications, Vol 33, 1999, 341-356.

[BCJLM97] A. Browne, E. Clarke, S. Jha, D. Long, W. Marrero, *An improved algorithm for the evaluation of fixpoint expressions*, **TCS 178** , 1997, 237-255.

[CANPW98] W. Charatonik, D. McAllester, D. Niwínski, A. Podelski, I. Walukiewicz, *The Horn Mu-calculus*, LICS, 1998, 58-69.

[CES86] E. M. Clarke, E. A. Emerson, A. P. Sistla, *Automatic Verification of finite-state concurrent systems using temporal logic specifications*, ACM TOPLAS, 8, 1986, 244-263.

[CS93] R. Cleaveland, B. Steffan, *A linear time model-checking algorithm for the alternation-free modal mu-calculus*, Formal methods in system design, 2 (1993), 121-148.

[E90] E. Emerson, *Temporal and modal logic*, Handbook of Theoretical Computer Science, 1990, 997-1072.

[E97] E. Emerson, *Model-Checking and the Mu-Calculus*, in Descriptive Complexity and Finite Models, N. Immerman and Ph. Kolaitis eds., American Mathematical Society, 1997.

[EL86] E. A. Emerson, C.L.Lei, *Efficient model-checking in fragments of the propositional μ-calculus*, In Proc. of 1rst Symposium on Logic in Computer Science, 1986, 267-278.

[FG03] E. Foustoucos, I. Guessarian, *Complexity of Monadic inf-datalog. Application to Temporal Logic*, Extended abstract in Proceedings 4th Panhellenic Logic Symposium, 2003, 95-99.

[GGV02] G. Gottlob, E. Grädel, H. Veith, *Datalog LITE: temporal versus deductive reasoning in verification*, ACM Trans. on Comput. Logic, 3, 2002, 39-74.

[GK02] G. Gottlob and C. Koch, *Monadic Datalog and the Expressive Power of Web Information Extraction Languages*, Proc. PODS'02, 17-28.

[GV04] A. Griffault and A. Vincent, *The Mec 5 model-checker*, CAV'04, Lecture Notes in Computer Science, 3114 , 2004, 488–491.

[GFAA03] I. Guessarian, E. Foustoucos, T. Andronikos, F. Afrati, *On Temporal Logic versus Datalog*, **TCS**, 303, 2003, 103-133.

[J00] M. Jurdzinski, *Small progress measures for solving parity games*, Proc. STACS'2000, 290-301.

[JPZ06] M. Jurdzinski, M. Paterson, U. Zwick,*A Deterministic Subexponential Algorithm for Solving Parity Games*, Proc. SODA 2006, 117-123.

[K83] D. Kozen, *Results on the propositional μ-calculus*, **TCS**, 27, 1983, 333-354.

[M95] A. Mader, *The modal μ-calculus, model-checking, equations systems and Gauss elimination*, TACAS 95, 1995, 44-57.

[P76] D. Park, *Finiteness is μ-ineffable*, **TCS**, 3, 1976, 173-181.

[V03] A. Vincent, *Conception et réalisation d'un vérificateur de modèles AltaRica*, PhD thesis, LaBRI, University of Bordeaux 1, 2003, http://altarica.labri.fr/Doc/Biblio/Author/VINCENT-A.html.

[S96] H. Seidl, *Fast and simple nested fixpoints*, **IPL**, 59 (6), 1996, 303-308.

Communicated by (The editor will be set by the publisher).
(The dates will be set by the publisher).