Complexity of Monadic inf-datalog. Application to Temporal Logic.

Eugénie Foustoucos* **Irène Guessarian****

Abstract: In [12] we defined Inf-Datalog and characterized the fragments of Monadic inf-Datalog that have the same expressive power as Modal Logic (resp. CTL, alternation-free Modal μ-calculus and Modal μ-calculus). We study here the time and space complexity of evaluation of Monadic inf-Datalog programs on finite models. We deduce a new unified proof that model checking has
1. linear data and program complexities (both in time and 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).

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 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 $\mathcal{K}$.

In [12] we introduced the language inf-Datalog, which extends usual least fixpoint semantics of Datalog with greatest fixpoint semantics, we gave translations from various temporal logics (CTL, ETL, alternation-free Modal μ-calculus, and Modal μ-calculus, by increasing order of expressive power [7]) into Monadic inf-Datalog and we also gave translations from fragments of Monadic inf-Datalog into these logics. In this paper we give upper bounds for evaluating Monadic inf-Datalog queries: we describe an algorithm evaluating Monadic 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 becomes linear when the program is stratified (with respect to least and greatest fixed points nesting): from this we derive a unified proof of the (known) linear-time data complexity of the model checking problem for CTL, ETL and the alternation-free μ-calculus. The program complexity of our algorithm is linear-time and linear-space, and the data complexity is linear-space too. Using then our translations in [12] between the temporal logic paradigm and the database paradigm, we can deduce upper bounds for the complexity of the model checking problem in the aforementioned temporal logics. This is worthwhile especially for the space complexity which is less studied than the time complexity.

* aflaw@otenet.gr, eugenie@math.uoa.gr, MPLA, Department of Mathematics, National and Capodistrian University of Athens, Panepistimiopolis, 15784 Athens, Greece.
** **Corresponding author: ig@liafa.jussieu.fr** LIAFA, UMR 7089, Université Paris 7, case 7014, 2 Place Jussieu, 75251 Paris Cedex 5, France.

2 Definitions

We first recall the basic definitions about Datalog.

Definition 1 *A Datalog program Π is a finite set of function-free Horn clauses, called rules, of the form:* $\varphi(x_1,\ldots,x_n) \longleftarrow \psi_1(y_{1,1},\ldots,y_{1,n_1}),\ldots,\psi_k(y_{k,1},\ldots,y_{k,n_k})$ *where:*

- $x_1,\ldots,x_n$ *are variables,*
- $y_{i,j}$*'s are either variables or constants,*
- φ *is a predicate, called the head of the rule, and*
- $\psi_1(y_{1,1},\ldots,y_{1,n_1}),\ldots,\psi_k(y_{k,1},\ldots,y_{k,n_k})$ *is the body of the rule,*
- *A predicate which appears on the head of some rule is called intensional database predicate (IDB predicate), whereas the rest are called extensional database predicates (EDB predicates),*
- ψ_i*'s are literals which are either of the form* $\varphi_i(y_{i,1},\ldots,y_{i,n_i})$ *or of the form* $\neg\varphi_i(y_{i,1},\ldots,y_{i,n_i})$*, with* φ_i *an EDB predicate, or an IDB predicate; if* ψ_i *is of the form* $\neg\varphi_i(y_{i,1},\ldots,y_{i,n_i})$*, then all rules with head* φ_i *have only EDB predicates in their bodies.*

The dependency graph of a Datalog 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 $\varphi \Longleftarrow \psi$).* Two predicates that belong to the same strongly connected component of the dependency graph are said to be* **mutually recursive**.

Remark 2 We should use the notation Datalog$^\neg$ for our language, because some negations may occur in rule bodies, but for simplicity sake, we will use only Datalog. Notice that we use negation in a very restrictive way (which reduces to using negation on EDB predicates only); our Datalog programs are always stratified with respect to negation, with at most two strata.

A Datalog program Π is said to be *monadic* if all the predicates occurring in the heads of the rules have arity at most one.

A *Datalog query* is a pair (Π,φ) consisting of a Datalog program Π together with one of its IDB predicates φ called *goal* predicate.

For any database D, $T_\Pi(D)$ is the set of ground facts which can be deduced from D by one single application of rules in Π, and $\varphi_\Pi(D)$ is the set of ground facts about φ which can be deduced from D by applications of the rules in Π; $\varphi_\Pi(D)$ is the set of **answers** to query (Π,φ) in D. The usual semantics of Datalog computes $\varphi_\Pi(D)$ by computing least fixed points only. Here, the semantics will be slightly different, because some IDB predicates will be "tagged" by an overline indicating that they must be computed with a greatest fixed point $\overline{\varphi_\Pi}(D)$ instead of a least fixed point $\varphi_\Pi(D)$.

We now define inf-Datalog.

Definition 3 *An inf-Datalog program is a Datalog program where some IDB predicates are tagged with an overline indicating that they must be computed as greatest fixed points, and where 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.*

An inf-Datalog program is said to be **monadic** *if all the predicates occurring in the heads of the rules 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.*

Our approach allows us to define some recursive predicates without initialization rules (non-recursive rules with this predicate in the head); such recursive predicates must be tagged. This approach is necessary in order to be able to express properties such as fairness (something must happen infinitely often).

The above notion of stratification is the natural counterpart (with respect to greatest fixed points) of the well-known stratification with respect to negation. We give an example of a stratified inf-Datalog program.

Example 4 Consider as database an infinite full binary tree, 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

* The direction of arrows is reversed with respect to the usual notion of dependency graph

some property of the nodes of the tree). The program P below, has as IDB predicates $\overline{\theta}$ (computed as a greatest fixed point) and φ (computed as a least fixed point)

$$P: \begin{cases} \overline{\theta}(x) \longleftarrow p(x), Suc_0(x,y), Suc_1(x,z), \overline{\theta}(y), \overline{\theta}(z) \\ \varphi(x) \longleftarrow \overline{\theta}(x) \\ \varphi(x) \longleftarrow Suc_0(x,y), Suc_1(x,z), \varphi(y), \varphi(z) \end{cases}$$

The IDB predicate $\overline{\theta}$ in this program implements the modality $\mathbf{A}Gp$ on the infinite full binary tree, and the IDB predicate φ implements the modality $\mathbf{A}F\mathbf{A}Gp$: $\mathbf{A}Gp$ 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 $\mathbf{A}\big(\top\mathbf{U}\mathbf{A}(\bot\widetilde{\mathbf{U}}p)\big)$. 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 basic definitions about the μ-calculus can be found in [2,8].)

3 Complexity of Monadic inf-Datalog

Theorem 4 *Let P be a stratified Monadic inf-Datalog program having I IDB symbols, and D 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 $n \times I$ and space $n \times I$.*

Proof. By induction on the number p of strata. Assume P has a single stratum, and, e.g. all IDBs are untagged, hence computed as least fixed points. Let $\varphi_1, \dots, \varphi_I$ be the IDBs, then the answer $f_1, \dots, f_I$ to the set of queries $(P, \varphi_1), \dots, (P, \varphi_I)$ defined by P is equal to $\sup_{n \in \mathbb{N}} T_P^n(\emptyset, \dots, \emptyset)$ and, because D has n objects only, this least upper bound is obtained after at most $n \times I$ steps. Same 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 $n \times I_j$, hence for the whole of P the complexity will be $n \times \sum_j I_j = n \times I$. The space complexity is clear too because we have at any time at most I IDBs true of at most n data objects.

This bound is tight as shown in the next example 6. □

Theorem 4 subsumes a result of [11], where it is shown that the data complexity of Monadic Datalog is linear-time; we prove that both the data complexity and the program complexity of stratified Monadic inf-Datalog are linear-time and linear-space: hence adding greatest fixed points in a stratified way increases the expressive power of Monadic Datalog without increasing its evaluation complexity. As a consequence of theorem 4 we get a new unified proof of the following result.

Corollary 5 *The model checking problem for CTL, ETL and alternation-free Modal μ-calculus can be solved in time and space $O(|M| \times |f|)$, where $|M|$ (resp. $|f|$) is the size of the model (resp. the formula); hence both the data and program complexities are linear in time and space.*

Proof. Indeed we give in [12] a translation from CTL, ETL and alternation-free Modal μ-calculus into Monadic stratified modal inf-Datalog such that the number of IDBs in the program is less than the size of the formula. □

A unified proof of the time-linearity wrt. the size of the model is also given in [10], and other proofs are given in [5,6].

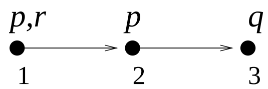

FIGURE 1 A data structure of size 3

EXAMPLE 6 Consider the structure given in figure 1, where $suc(1,2), suc(2,3), p(1), p(2), q(3), r(1)$ hold and the Monadic Datalog program:

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

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

We now turn to Monadic inf-Datalog programs with alternations.

Definition 6 *The syntactic alternation depth of P is the largest k such that there exists in the dependency graph of P a cycle $\varphi_1 \longrightarrow \varphi_{1,2} \longrightarrow \cdots \varphi_{1,n_1} \longrightarrow \overline{\varphi_2} \longrightarrow \overline{\varphi_{2,2}} \longrightarrow \cdots \overline{\varphi_{2,n_2}} \longrightarrow \cdots \longrightarrow \varphi_{k-1} \longrightarrow \varphi_{k-1,2} \longrightarrow \cdots \varphi_{k-1,n_{k-1}} \longrightarrow \overline{\varphi_k} \longrightarrow \overline{\varphi_{k,2}} \longrightarrow \cdots \overline{\varphi_{k,n_k}} \Longrightarrow \varphi_1$ having k alternations of k pairwise distinct tagged/untagged IDBs $\varphi_1, \overline{\varphi_2}, \ldots, \overline{\varphi_k}$ depending on each other (we will often omit the overlines for simplicity it will be understood that whenever φ_i is overlined, φ_{i-1} is not overlined and vice-versa).*

The previous definition of alternation depth is similar to the syntactic ad as defined in [3,14] and different from the notion of alternation depth of [2,9].

EXAMPLE 7 Consider the inf-Datalog programs P_φ and P_ψ respectively corresponding to the μ-calculus formulas $\varphi \equiv \mu X.\nu Y.(X \vee Y \vee \mu Z.\nu W.(X \vee Z \vee (p \wedge W)))$ and $\psi \equiv \mu Y.\nu X.(X \vee Y \vee \mu Z.\nu W.(X \vee Z \vee (p \wedge W)))$:

$$P_\varphi \left\{ \begin{array}{lcl lcl} \overline{W1}(x) & \longleftarrow & p(x), \overline{W1}(x) & \overline{Y3}(x) & \longleftarrow & Z2(x) \\ \overline{W1}(x) & \longleftarrow & X4(x) & \overline{Y3}(x) & \longleftarrow & \overline{Y3}(x) \\ \overline{W1}(x) & \longleftarrow & Z2(x) & \overline{Y3}(x) & \longleftarrow & X4(x) \\ Z2(x) & \longleftarrow & \overline{W1}(x) & X4(x) & \longleftarrow & \overline{Y3}(x) \end{array} \right. \qquad P_\psi \left\{ \begin{array}{lcl lcl} \overline{W1}(x) & \longleftarrow & p(x), \overline{W1}(x) & \overline{X3}(x) & \longleftarrow & Z2(x) \\ \overline{W1}(x) & \longleftarrow & \overline{X3}(x) & \overline{X3}(x) & \longleftarrow & Y4(x) \\ \overline{W1}(x) & \longleftarrow & Z2(x) & \overline{X3}(x) & \longleftarrow & \overline{X3}(x) \\ Z2(x) & \longleftarrow & \overline{W1}(x) & Y4(x) & \longleftarrow & \overline{X3}(x) \end{array} \right.$$

Both P_φ and P_ψ have syntactic alternation depth 4, and their dependency graphs are pictured below, we will see later that their semantic alternation depth differ.

FIGURE 2 Dependency graph of P_φ and P_ψ

Theorem 7 *Let P be a program with syntactic alternation depth k fixed. Assume P has I mutually recursive IDBs. 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 $O\big((n+1)^k \times I\big)$ and space $O(n \times I)$.*

Proof. The proof proceeds in three cases.

Case 1. For simplicity, we will first study the case when $I = k$, k even: 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 $\overline{\varphi_2}$, ..., 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. Then $P = P_k$ has the following form:

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

The idea of the algorithm is obtained by adapting an algorithm given in [2] for evaluating boolean μ-calculus formulas and proceeds as follows. Let $f_1, \ldots, f_k$ be 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 P'_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 3.

Notice that in the $k-1$ first nested loops the indices have to go from 1 to $n+1$: indeed each individual f_j is computed in at most n steps* , but then we have to substitute the value just computed for f_j in $f_1, \ldots, f_{j-1}$ whence the need for one more round of iterations. At the end $f_1, \ldots, f_k$ contain the answers to the queries defined by $\varphi_1, \ldots, \overline{\varphi_k}$. The complexity of the algorithm is $n + (n+1)^2 + \cdots + (n+1)^{k-1} + n(n+1)^{k-1}$ which is $O((n+1)^k) < O((n+1)^k \times I)$. The algorithm for $I = k$, k odd, is similar. The formal proof consists in proving the slightly more general lemma 8 below.

```
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);
            ENDFOR (j_1)
            IF j_2 ≤ n THEN f_2  := T_{P'_2}  (f_1, f_2, ..., 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 3 Algorithm1

Case 2. The generalization to the case when P has I mutually recursive IDBs, $I > k$, is straightforward: let the IDBs of P be for instance $\Phi = \Phi_1 \cup \overline{\Phi_2} \cup \Phi_3 \cup \cdots \cup \overline{\Phi_k}$. All the IDBs in Φ_i (resp. $\overline{\Phi_j}$) are untagged (resp. tagged). 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 $\overline{\Phi_k}$ are computed as greatest fixed points. Assume that, for $i = 1, \ldots, k$, Φ_i has m_i IDBs $\varphi_{i,1}, \ldots, \varphi_{i,m_i}$.
Then it suffices to substitute for instruction: $f_i := T_{P'_i}(f_1, f_2, \ldots, f_i, \ldots, f_k)$ the set of m_i instructions:

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

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 $\varphi_{1,1}, \ldots, \varphi_{1,m_1} \ldots . \varphi_{k,1}, \ldots, \varphi_{k,m_k}$. Now the complexity of the algorithm becomes:
$n \times m_k + (n+1)^2 \times m_{k-1} + \cdots + (n+1)^{k-1} \times m_2 + n(n+1)^{k-1} \times m_1 < (n+1)^k \times \left(\sum_{i=1}^k m_i\right) = O\left((n+1)^k \times I\right)$.
Case 3. Last, in the most general case, the rules of P can be partitioned into n disjoint sets $\Pi_0, \Pi_1, \ldots, \Pi_n$: Π_0 is a stratified program (possibly empty), each $\Pi_i, i = 1, \ldots, n$, has I_i mutually recursive IDBs with k_i alternations of least fixed points and greatest fixed points, and J_i IDBs which are defined in a stratified way (we will nickname them "stratified" IDBs); the I_i mutually recursive IDBs can depend on the IDBs of $\Pi_j, j < i$, the J_i "stratified" IDBs can depend both on the I_i mutually recursive IDBs and on the IDBs of $\Pi_j, j < i$, and none of the IDBs of Π_i can depend on the IDBs of $\Pi_j, j > i$. Then we evaluate first the queries defined by the IDBs of Π_0 in time $O(n \times J_0)$ (as in theorem 4), then the queries defined by the I_1 mutually recursive IDBs of Π_1 in

* Notice that each instruction $f_i := T_{P'_i}(f_1, f_2, \ldots, f_i, \ldots, f_k)$ is counted for one step; indeed this can be achieved with a careful implementation [1].

time $O((n+1)^{k_1} \times I_1)$ as in case 2, then the queries defined by the J_1 "stratified" IDBs of Π_1 in time $O(n \times J_1)$ as in theorem 4, etc. Finally, the total time complexity is at most $O((n+1)^{\max k_i} \times (\sum_{i=1}^{n} I_i) + n \times (\sum_{i=0}^{n} J_i)) \leq O((n+1)^k \times I)$ where $k = \max\{k_i | 1 \leq i \leq n\}$.

In all three cases, the space complexities of the algorithms which are used 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 8 *Let P be a program with syntactic alternation depth 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 in time at most $(n+1)^k - 1$.*

Proof. By induction on k. Assume P has exactly k mutually recursive IDBs and $k-1$ alternations of least fixed points and greatest fixed points; the first IDB is assumed to be a least fixed point, but the other cases would be similar.

Basis. For $k = 1$, the lemma is clear: there is no alternation, and the time complexity is $n = (n+1)^1 - 1$.

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

$$P_{k+1}(g_1, \ldots, g_p) \begin{cases} P'_{k+1} \begin{cases} \varphi_{k+1}(x) \longleftarrow \cdots \\ \qquad\vdots \\ \varphi_{k+1}(x) \longleftarrow \cdots \end{cases} \\ P_k(g_1, \ldots, g_p, \varphi_{k+1}) \end{cases}$$

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 $P_k(g_1, \ldots, g_p, \varphi_{k+1})$ (with parameters $g_1, \ldots, g_p$ *and* φ_{k+1}). 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 definition of the 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, $f^1_{k+1} = T_{P'_{k+1}}(f^0_{k+1}) = T_{P'_{k+1}}(\emptyset) = T_{P'_{k+1}}[\emptyset/\varphi_{k+1}, f_k(\emptyset)/\overline{\varphi_k}, \ldots, f_1(\emptyset)/\varphi_1]$; $f_k(\emptyset), \ldots, f_1(\emptyset)$ are the answers to the queries $\varphi_k, \ldots, \varphi_1$ defined by the program $P_k(g_1, \ldots, g_p, \emptyset)$ with $k-1$ alternations; hence by the induction hypothesis, $f_k(\emptyset), \ldots, f_1(\emptyset)$ are computed in time at most $(n+1)^k - 1$ by algorithm1 (for k), and $T_{P'_{k+1}}[\emptyset/\varphi_{k+1}, f_k(\emptyset)/\overline{\varphi_k}, \ldots, f_1(\emptyset)/\varphi_1]$ is thus computed in time at most $(n+1)^k - 1 + 1 = (n+1)^k$. Similarly, for $j = 1, \ldots, n-1$, each of $f^{j+1}_{k+1} = T_{P'_{k+1}}[f^j_{k+1}/\varphi_{k+1}, f_k(f^j_{k+1})/\overline{\varphi_k}, \ldots, f_1(f^j_{k+1})/\varphi_1]$ is computed in time at most $(n+1)^k$. As explained above, f^n_{k+1} is the answer f_{k+1} to the query φ_{k+1} defined by the program $P_{k+1}(g_1, \ldots, g_p)$; therefore, the answers $f_k, \ldots, f_1$ to the queries $\varphi_k, \ldots, \varphi_1$ 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 in time at most $(n+1)^k - 1$, by the induction hypothesis; hence a total time $n(n+1)^k + (n+1)^k - 1 = (n+1)^{k+1} - 1$. □

We can restate theorem 7 as: algorithm1 computes the answers to queries defined by Monadic inf-Datalog programs with linear-space (data and program) complexities, linear-time program complexity, and polynomial-time data complexity.

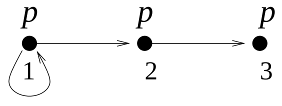

FIGURE 4 Another structure of size 3

EXAMPLE 9 Consider the structure given in figure 4, where $suc_1(1,1), suc_0(1,2), suc_0(2,3), p(1), p(2), p(3)$ hold and the program P below (where $I = 2 = k$):

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

Then algorithm1 will compute: 1. for $f_2 = \top$, $f_1 = \emptyset, f_1 = \{1,2\}$, and $f_2 = \{1,2\}$; then, 2. for $f_2 = \{1,2\}$, $f_1 = \emptyset, f_1 = \{1\}$, and $f_2 = \{1\}$; then, 3. for $f_2 = \{1\}$, $f_1 = \emptyset, f_1 = \{1\}$, 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.)

In [9] a finer notion of alternation depth is defined. Our algorithm1 can be improved to match this finer notion by avoiding some useless recomputations of fixed points. To this end we need to define a notion of semantic alternation depth corresponding to the definition of [9]. For simplicity we will detail only the case when the number of IDBs is equal to the syntactic alternation depth. The other cases can be treated similarly but at the cost of heavier notation.

Definition 9 *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 $\overline{\varphi_k} \Longrightarrow \varphi_1 \longrightarrow \overline{\varphi_2} \longrightarrow \cdots \longrightarrow \varphi_{k-1} \longrightarrow \overline{\varphi_k}$ in the dependency graph of P. The semantic alternation depth of φ_i, denoted by $ad(\varphi_i)$, is defined as follows: $ad(\varphi_1) = 1$ and $ad(\varphi_{i+1}) = ad(\overline{\varphi_i}) + e$ where*

$$e = \begin{cases} 1, & \text{if there is a path from } \varphi_{i+1} \text{ to } \overline{\varphi_i}, \text{ in which appears no } \varphi_s,\ s > i+1, \\ & \text{in the dependency graph of } P, \\ 0, & \text{otherwise} \end{cases}$$

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).

We will show that the above definition of semantic alternation depth coincides with the a.d. as it is defined in [9]. To prove this we need the following proposition.

Proposition 10 *Let (P, φ_k) be the inf-Datalog query corresponding to formula $\psi = \nu X_k \ldots \mu X_{k-1} \ldots \nu X_2$ $\mu X_1 . f(X_1, X_2, \ldots, X_{k-1}, X_k)$, assume the IDB φ_i stands for the variable X_i. Let $i < j$, in ψ, X_j occurs free in the scope $\mu X_i \ldots \nu X_{i-1} \ldots \nu X_2 \mu X_1 . f(X_1, X_2, \ldots, X_{k-1}, X_k)$ of μX_i (resp. νX_i) , if and only if there exists in the dependency graph of P a path $\varphi_j \to \cdots \to \varphi_i$ that does not contain any φ_s, $s > j$.*

Example 11 Let φ and ψ be as in example 7. In P_φ $ad(P_\varphi) = 3 = ad(G^4_X)$ and $ad(\overline{G^3_Y}) = ad(G^2_Z) = 2$, while in P_ψ $ad(P_\psi) = 4 = ad(G^4_Y)$ and $ad(\overline{G^3_X}) = 3$, $ad(G^2_Z) = 2$.

Lemma 11 *Consider a modal μ-calculus formula ψ of the form $\psi = \nu X_k \ldots \mu X_{k-1} \ldots \nu X_2 \mu X_1 . f(X_1, X_2, \ldots,$ $X_{k-1}, X_k)$ and let $(P(\psi), \overline{\varphi_k})$ be the inf-Datalog translation of ψ; (i) the syntactic alternation depth of $P(\psi)$ coincides with the simple (or syntactic) alternation depth of ψ as it is defined in [3], and (ii) the semantic alternation depth of $P(\psi)$ coincides with the alternation depth of ψ as it is defined in [9].*

Proof. (i) and (ii) follow from our definitions of alternation depth, from the definitions of alternation depth of ψ in [3,9], and from proposition 10. □

Theorem 12 *Let P be a program with syntactic alternation depth k, 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 $g_1, \ldots, g_p$, we can compute the answers $f_1, \ldots, f_k$ to the queries $\varphi_1, \ldots, \overline{\varphi_k}$ on D in time $O(n^d)$ where $d = ad(P)$ is the semantic alternation depth of P.*

Proof. The proof is similar to the proof of theorem 7 (Case1), but uses a slight improvement of Algorithm1, which takes into account the fact that some iterations computing the f_is need not be nested (basically it avoids recomputing f_is when their value does not change). We just need to modify lemma 8 into lemma 14, and theorem 12 will follow. □

The notion of semantic a.d. can be generalized to any form of monadic inf-Datalog program P, hence theorem 12 can be extended to programs belonging to cases 2 and 3 of theorem 7.

Corollary 13 *1. The set of queries defined by Monadic inf-Datalog programs can be computed in time polynomial in the size of the data structure, exponential in the number of alternations of least fixed points and greatest fixed points, and linear in the number of IDBs. The space complexity is linear in $n \times I$ (n is the size of the structure and I the number of IDBs).*
2. The model checking problem for the Modal μ-calculus can be solved in time polynomial in the size of the model and exponential in the semantic alternation depth of the formula. The space complexity is linear in $|M| \times |f|$ ($|M|$ is the size of the model and $|f|$ the size of the formula).

Proof. Theorem 7 states that the Monadic inf-Datalog queries of alternation depth k can be computed in time $O\big((n+1)^k \times I\big)$ and space $O(n \times I)$, whence case 1 of the corollary.

2. In [12] we translated a modal μ-calculus formula f into a Monadic (in fact modal) inf-Datalog program P, such that (1) the structure (M, s) is a model of f if and only if s is an answer to one of the queries defined by P on M, and (2) the number I of IDBs is less than the size of formula f. By lemma 11 the semantic alternation depth of P is equal to $ad(f)$ as defined in [9]; theorem 12 then implies that the time complexity of model checking for the Modal μ-calculus is in $O((|M|+1)^k \times |f|)$, whence 2. □

Lemma 14 *Let P be a program with syntactic alternation depth k, IDBs $\varphi_1, \overline{\varphi_2}, \ldots, \overline{\varphi_k}$, $k \geq 1$ and with parameters $g_1, \ldots, g_p$. Let D be a database with n elements. For any given values of the parameters $g_1, \ldots, g_p$ and of $\varphi_{j+1}, \ldots, \varphi_k$, (if $k > j \geq 1$), we can compute the answers $f_1(g_1, \ldots, g_p, \varphi_{j+1}, \ldots, \varphi_k), \ldots, f_j(g_1, \ldots, g_p, \varphi_{j+1}, \ldots, \varphi_k)$ to the queries $\phi_1, \ldots, \phi_j$ (defined by $P_j(g_1, \ldots, g_p, \varphi_{j+1}, \ldots, \varphi_k)$) in time at most $O(n^{ad(\varphi_j)})$.*

Proof. By induction on (k, j), ordered lexicographically.

Basis: If $k = 1$, then $j = 1$, $ad(\varphi_1) = 1$ there is no alternation and we compute $f_1(g_1, \ldots, g_p)$ as in lemma 8 (Basis).

If $k \neq 1$, and $j = 1$, again $ad(\varphi_1) = 1$ 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 n steps.

Induction: Assume the result is true for (k, j) and prove it for $(k, j+1)$. Then we distinguish two cases:

case (i): there is an edge from φ_{j+1} to some φ_i, $i < j+1$; hence there is a path from φ_{j+1} to φ_j ($\varphi_{j+1} \longrightarrow \varphi_i \longrightarrow \varphi_{i+1} \cdots \varphi_{j-1} \longrightarrow \varphi_j$), in which appears no φ_s, $s > j+1$, and $ad(\varphi_{j+1}) = ad(\varphi_j)+1$; the IDB φ_j thus depends on φ_{j+1}. This means that $f_j(g_1, \ldots, g_p, f^q_{j+1}, \varphi_{j+2}, \ldots, \varphi_k)$ is *apriori* different from $f_j(g_1, \ldots, g_p f^{q+1}_{j+1}, \varphi_{j+2}, \ldots, \varphi_k)$, therefore the `FOR` j_{j+1} loop of algorithm1 needs to be nested over the loops computing $f_1, \ldots, f_j$; as by the induction hypothesis the latter are computed in time $O(n^{ad(\varphi_j)})$, together with the final englobing loop `FOR` j_{j+1}, $f_1(g_1, \ldots, g_p, \varphi_{j+2}, \ldots, \varphi_k), \ldots, f_{j+1}(g_1, \ldots, g_p, \varphi_{j+2}, \ldots, \varphi_k)$ will be computed in time $O(n^{ad(\varphi_j)} \times n) = O(n^{ad(\varphi_j)+1}) = O(n^{ad(\varphi_{j+1})})$.

case (ii): there is no edge from φ_{j+1} to some φ_i, $i < j+1$; then any path from φ_{j+1} to φ_j, goes through some φ_l, $l > j+1$ (otherwise the first edge of a such path $\varphi_{j+1} \Longrightarrow \varphi_j$ would go 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}: this means that, for $i \leq j$, all the $f_i(g_1, \ldots, g_p, f^q_{j+1}, \varphi_{j+2}, \ldots, \varphi_k)$ are *apriori* equal to the $f_i(g_1, \ldots, g_p, f^{q+1}_{j+1}, \varphi_{j+2}, \ldots, \varphi_k)$, therefore the `FOR` j_{j+1} loop of algorithm1 does not need to be nested over the loops computing $f_1, \ldots, f_j$, and can just be performed after completion of these loops, in time $O(n)$; as by the induction hypothesis the latter are computed in time $O(n^{ad(\varphi_j)})$, the time for computing $f_1, \ldots, f_j, f_{j+1}$ will be $O(n^{ad(\varphi_j)} + n) = O(n^{ad(\varphi_j)}) = O(n^{ad(\varphi_{j+1})})$. □

4 Conclusion

We gave a (linear-) polynomial-time algorithm computing the answers to the queries defined by a (stratified) Monadic inf-Datalog program. The time complexity of this algorithm is $O(n^k \times I)$ where n is the size of the database, I the number of IDBs and k the semantic alternation depth. For model checking formulas of $L\mu_k$ (which is equivalent to a fragment of Monadic inf-Datalog), [8] describes a semi-naive algorithm avoiding some redundant computations and running in time $O\big((n \times |f|)^k\big)$, whilst we obtain $O\big((n+1)^k \times |f|\big)$ by combining theorem 12 and our translation from the modal μ-calculus into Monadic inf-Datalog [12]. We believe that this bound could be slightly improved: indeed [4] gives an algorithm model checking formulas of $L\mu_k$ running in time $O\big((n \times |f|)^{2+k/2}\big)$; however the space complexity of the improved algorithm in [4] is exponential whilst the space complexity of the (semi-)naive algorithms is polynomial [8]. By reducing the problem to parity games, [13] gives an algorithm running in time $O\big((n \times |f|)^{2+k/2}\big)$ and polynomial space.

Acknowledgments: The second author is very grateful to Damian Niwiński for enlightening discussions.

5 References

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

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

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

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

[5] 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.

[6] 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.

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

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

[9] 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.

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

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

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

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

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