Complexity of Monadic inf-datalog. Application to temporal logic.

Eugénie Foustoucos, Irene Guessarian

Eugénie Foustoucos Irène Guessarian

Contact Author Irène Guessarian

Address: LIAFA, Université Paris 7, case 7014, 2 Place Jussieu, 75251 Paris Cedex 5, France.

email: **ig@liafa.jussieu.fr**

Classification of paper: **Logic in Computer Science**

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

Eugénie Foustoucos* **Irène Guessarian****

Abstract: In [] 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 [] 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 []) 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 [] 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

The basic definitions about Datalog can be found in [,], and the basic definitions about the μ-calculus can be found in [,]. We proceed directly with the definition of inf-Datalog.

Definition 1 *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 2 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 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)

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

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 $\mu\varphi.\Big(\nu\theta.(p \wedge \mathbf{A}\circ\ \theta) \bigvee \mathbf{A}\circ\ \varphi\Big)$.

3 Complexity of Monadic inf-Datalog

Theorem 2 *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 (of the form (P,φ), where φ is an IDB of P) can be evaluated 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 $\varphi_1, \ldots, \varphi_I$ 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 $\sup_{n\in\mathbb{N}} T_P^n(\emptyset, \ldots, \emptyset)$ 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 ?? . □

Theorem ?? subsumes a result of [], 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 ?? we get a new unified proof of the following result.

Corollary 3 *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 [] 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 [], and other proofs are given in [,].

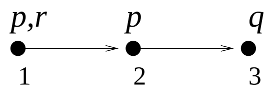

FIGURE 1 A data structure of size 3

EXAMPLE 4 Consider the structure given in ?? , where $suc(1,2), suc(2,3), p(1), p(2), q(3), r(1)$ hold and the Monadic Datalog program:

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

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.

Theorem 4 *Let P be a program with $k-1$ alternations of least fixed points and greatest fixed points (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 in time $O\big((n+1)^k \times I\big)$ and space $O(n \times I)$.*

Proof. Program P has $k-1$ alternations of least fixed points and greatest fixed points, which means that there exist IDBs $\varphi_1, \overline{\varphi_2}, \dots, \varphi_{k-1}, \overline{\varphi_k}$, computed in the order: first φ_1, then $\overline{\varphi_2}$, …, and last $\overline{\varphi_k}$. For simplicity, we first assume that $I = k$, k even, then $P = P_k$ has the following form:

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

The idea of the algorithm is obtained by adapting an algorithm given in [] for evaluating boolean μ-calculus formulas and proceeds as follows. Let $f_1, \dots, f_k$ be the queries defined by $\varphi_1, \dots, \overline{\varphi_k}$. 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 ?? .

```
ALGORITHM1
VAR $j_1,\dots,j_k$:  indices;
$f_k$ := $\top$;
FOR $j_k = 1$ TO $n+1$ DO
   $f_{k-1}$ := $\emptyset$;
   FOR $j_{k-1} = 1$ TO $n+1$ DO
      $f_{k-2}$ := $\top$;
      ⋮
         $f_2$ := $\top$;
         FOR $j_2 = 1$ TO $n+1$ DO
            $f_1$ := $\emptyset$;
            FOR $j_1 = 1$ TO $n$ DO
               $f_1$:= $T_{P_1}$ $(f_1, f_2, \dots, f_k)$;
            ENDFOR $(j_1)$
            $f_2$ := $T_{P'_2}$ $(f_1, f_2, \dots, f_k)$;
         ENDFOR $(j_2)$
      ⋮
      $f_{k-1}$ := $T_{P'_{k-1}}(f_1, f_2, \dots, f_{k-1}, f_k)$;
   ENDFOR $(j_{k-1})$
   $f_k$ := $T_{P'_k}(f_1, f_2, \dots, f_{k-1}, f_k)$;
ENDFOR $(j_k)$
```

FIGURE 2 Algorithm1

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 $f_1, \dots, f_{j-1}$ whence the need for one more round of iterations. At the end $f_1, \dots, f_k$ contain the answers to the queries defined by $\varphi_1, \dots, \overline{\varphi_k}$. The complexity of the algorithm is $(n+1)+(n+1)^2+\cdots+(n+1)^{k-1}+n(n+1)^{k-1}$ which is $O((n+1)^k)$.

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 Φ_k are computed as greatest fixed points. Assume Φ_i has m_i IDBs, for $i = 1, \dots, k$.

Then it suffices to substitute for instruction: $f_i := T_{P'_i}(f_1, f_2, \dots, f_i, \dots, f_k)$ the set of m_i instructions:

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

$$\vdots$$

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

where $T_{P'_i,l}(f_1, f_2, \dots, f_{i-1}, \dots, f_I)$ denotes the set of immediate consequences which can be deduced using the rules of P'_i with head $\varphi_{i,l}$. Now the complexity of the algorithm becomes:
$(n+1)\times m_k+(n+1)^2\times m_{k-1}+\cdots+(n+1)^{k-1}\times m_2+n(n+1)^{k-1}\times m_1$ which is an $O\big((n+1)^k \times \max\{m_i/i = 1, \dots, k\}\big) \leq O((n+1)^k \times I)$. □

We can restate ?? 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.

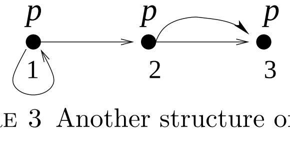

FIGURE 3 Another structure of size 3

EXAMPLE 5 Consider the structure given in ?? , 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 $I = k = 2$):

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

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.)

Corollary 5 *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 number of syntactic alternations 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. 2 follows from the fact that in [] we gave a translation from modal μ-calculus formulas into Monadic (in fact modal) inf-Datalog programs, such that the number k of alternations in the program is equal to the number of syntactic alternations [] of the formula and the number I of IDBs is less than the size of the formula. 1 is a restatement of ?? . □

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 $O(n^{k+1})$ where n is the size of the database and k the number of alternations. We believe that this bound could be slightly improved: indeed there are algorithms for model checking formulas of $L\mu_k$ (which is equivalent to a fragment of Monadic inf-Datalog), with upper bounds $O\big((n \times |f|)^k\big)$ [,] and $O\big((n \times |f|)^{2+k/2}\big)$ [] (compared to our bound $O\big((n+1)^{k+1} \times |f|\big)$); however the space complexity of the improved algorithm in [] becomes exponential whilst the space complexity of the naive algorithms is polynomial [].

5 References

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

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

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

[4] 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.

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

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

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

[8] 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.

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

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

[11] I. Guessarian, E. Foustoucos, T. Andronikos, F. Afrati, *On Temporal Logic versus Datalog*, to appear in **TCS**, available from http://www.liafa.jussieu.fr/~ig/gfaa.ps