On the Alternation-Free Horn μ-Calculus

Jean-Marc Talbot

Max-Planck Institut für Informatik
Stuhlsatzenhausweg 85 - 66123 Saarbrücken - Germany

Abstract. The Horn μ-calculus is a formalism extending logic programs by specifying for each predicate symbol with which (greatest or least) fix-point semantics, its denotation has to be computed. When restricted to a particular class of logic programs called uniform, the Horn μ-calculus provides a syntactic extension for Rabin tree automata. However, it has been shown [1] that the denotation of the Horn μ-calculus restricted to a uniform program remains a regular set of trees and that moreover, the emptiness of the denotation of a predicate p is a DEXPTIME-complete problem (in the size of the program). In [3], these results have been extended to uniform programs that may contain both existential and universal quantifications on the variables occurring in the body of "clauses": considering this extension, the denotation of a program remains a regular set of trees, but the best known algorithm for testing the emptiness of the denotation of a predicate is doubly-exponential in the size of the program.
In this paper, we consider uniform logic programs with both kinds of quantification in the body. But we add to the Horn μ-calculus a limitation on the way the fix-point semantics is specified for predicates. This restriction is close to the one defining the alternation-free fragment of the μ-calculus. Therefore, we name this fragment of the Horn μ-calculus the *alternation-free* fragment. We devise for it an algorithm which performs the emptiness test for the denotation of a predicate in single-exponential time in the size of the program.
To obtain this result, we develop a constructive approach based on a new kind of tree automata running on finite and infinite trees, called *monotonous tree automata*. These automata are defined by means of a family of finite and complete lattices. The acceptance condition for monotonous tree automata is based on the ordering relations of the lattices.

1 Introduction

The Horn μ-calculus [1] is a formalism extending logic programs by means of fix-point operators. In the classical framework of logic programming, the ground [1] semantics of a program is usually expressed in terms of a least fix-point computation. However, for some different purposes, the semantics related to the greatest fix-point (also called reactive semantics) may also be of some interest [11]. The idea of the Horn μ-calculus is to integrate this semantical point in the program itself by specifying for each predicate symbol whether its semantics has to be computed as a least or a greatest fix-point.

[1] In this paper, we consider the semantics of programs over the complete Herbrand base, that is atoms built over finite and infinite trees.

M. Parigot and A. Voronkov (Eds.): LPAR 2000, LNAI 1955, pp. 418–435, 2000.

Semantics for a logic program can alternatively be expressed in terms of ground proof trees. A ground atom belongs to the least fix-point semantics if there exists a finite ground proof tree rooted by this atom. Since the number of predicate symbols is finite, this finiteness condition over the proof tree can be rephrased as "for any predicate p, along every branch, one encounters only finitely many atoms defined with p". On the other hand, an atom belongs to the greatest fix-point semantics if it exists a ground (possibly infinite) proof tree rooted by this atom. Thus, in this case predicate symbols can occur freely infinitely often in atoms along each branch. The Horn μ-calculus captures such notions by associating with each predicate symbol a positive integer, called priority. An atom is then "accepted" by the program if there exists a proof tree for it such that the priorities of predicates in the atoms along each branch satisfy the *parity condition* [7]: the maximal priority occurring infinitely often along each branch is even. Roughly speaking, the semantics for predicate symbols with an odd arity is computed as a least fix-point and a greatest fix-point is used for predicate symbols with an even arity.
When this Horn μ-calculus is restricted to a class of logic programs called uniform programs [8], then its semantics coincide with the notion of regularity *à la* Rabin [14]: the set of grounds atoms accepted by the program is a regular set of trees. This result has extended other results concerning the least [108108] and the greatest [2626] fix-point semantics for uniform logic programs obtained in the area of set-based analysis. Hence, the Horn μ-calculus restricted to uniform programs can be viewed as a particular technique for performing (set-based) relaxation of the Horn μ-calculus for arbitrary programs.

As we have said above, uniform logic programs are related to set-based analysis; they are more specifically connected with some classes of set constraints called the definite [9] and the co-definite class [4]. To be fully precise, uniform logic programs correspond to the definite and the co-definite classes extended with some set description called quantified set expressions in [10] and membership expressions in [5656]. Quantified set expressions (or equivalently, membership expressions) are in fact intentional description of sets of the form $\{x|\Psi(x)\}$; they are interpreted as the set of all trees τ which satisfy the given property Ψ, *i.e.* as the set of trees τ such that $\Psi(\tau)$ holds. This property Ψ is presented as an existentially quantified conjunction of atoms. So, rephrased in the logic programming framework, such an expression would correspond to the clause $p(x) \leftarrow \Psi(x)$. In [16181618], the membership expressions have been extended by allowing variables to be also universally quantified. The aim of this extension was to provide a uniform view of the definite and the co-definite classes of set constraints. So, one may wonder whether this extension can be carried over the Horn μ-calculus for uniform programs. This would lead to consider "clauses"[2] of the from $p(t) \leftarrow \Psi$ where the local variables in Ψ (the ones which do not occur in $p(t)$) can be quantified existentially or universally.
The Horn μ-calculus restricted to uniform programs with both quantification kinds in the body has been considered by Charatonik, Niwiński and Podelski in [3] for a model-checking purpose. It is shown in that paper that this extension preserves the regularity,

[2] The word *clause* is used here in a not fully proper way: universal quantification on variables in the body part leads to formulas that are no longer Horn clauses.

in the sense that the denotation of the program is a regular set of trees. Furthermore, Charatonik and *al.* have proposed an algorithm performing the emptiness test for the denotation of some predicate; this algorithm runs in doubly-exponential time in the size of the program, whereas the best known lower-bound for the complexity of this problem is DEXPTIME.

As in [3], this paper is based on logic programs for which variables occurring in the body of clauses can be quantified universally or existentially together with a function that assigns to predicate symbols a priority (*i.e.* a natural number). However, we will consider only a fragment of this calculus; this fragment is defined by a syntactic restriction based on priorities. It is quite similar to the restriction defining from the μ-calculus its alternation-free fragment. Therefore, we call this fragment the *alternation-free Horn μ-calculus*. Roughly speaking the restriction for the alternation-free μ-calculus requires that the computation of the semantics of a formula can be achieved layer-by-layer, each of those layers representing an homogeneous block of fix-point quantifiers. In a similar way, for the alternation-free Horn μ-calculus, the semantics of any predicate p can not depend on the semantics of some predicate q having a priority strictly greater than the one of p: the denotation of a predicate can not depend on the denotation of predicates belonging to upper layers. For this alternation-free fragment, we rephrase its semantics by means of fix-point operators based on this idea of layers.

We show here that for the alternation-free Horn μ-calculus restricted to uniform programs with both quantification kinds in bodies, deciding whether the denotation of a predicate p is empty (that is, deciding whether there exists a tree τ such that $p(\tau)$ belongs to the denotation of the program) can be achieved in single-exponential time in the size of the program improving the result from [3] for this specific fragment. To obtain this result, we have designed a new kind of tree automaton, called *monotonous tree automaton*.

Those automata are based on a family of finite and complete lattices $(S_i, \preceq^i)$. A state is a tuple of the S_i's. The notion of runs coincides with the classical one but the acceptance condition of those runs is based on the family of orderings $(\preceq^0, \ldots, \preceq^k)$ on which lattices are defined. We show that for a monotonous tree automaton the emptiness of the accepted language can be checked in polynomial time in the size of the automaton. For a program from the alternation-free Horn μ-calculus (P, Ω), we fix a general shape of the monotonous automata we have to consider. Then, we design some fix-point operators defined over those automata that mimic the fix-point semantics of (P, Ω). Our algorithm yields a monotonous tree automaton that recognizes the semantics of (P, Ω).

The paper is organized as follows: in Section 2, we give the syntactic definition for the Horn μ-calculus as well as its semantics in terms of proof trees. We also present there the alternation-free fragment together with its alternative semantics in terms of fix-points. Section 3 is devoted to the presentation of the generalized uniform programs we consider. Those programs allow in *projection clauses* variables occurring in the body to be either universally or existentially quantified. This section settles also the main result of this paper that is the emptiness problem for the alternation-free Horn μ-calculus when restricted to those so-extended uniform programs is DEXPTIME-complete. The rest of the paper that is Section 4 addresses the method we used to obtain this result: we present there monotonous tree automata and give some basic results about them like the

regularity of the recognized language and the complexity for the emptiness problem. Then, we present an algorithm that given a uniform program from the alternation-free Horn μ-calculus computes an equivalent monotonous tree automaton: the semantics of program and the language accepted by the automaton coincide.

Due to lack of space, most of the proofs are not in this paper but can be found in the long version [17].

2 The Horn μ-Calculus

We consider a finite ranked signature Σ. We denote T^*_Σ, the set of all finite and infinite trees generated over Σ. For a tree τ, $dom(\tau)$ will denote its tree domain; for any position d in $dom(\tau)$, $\tau(d)$ denotes the function symbol from Σ labeling τ at position d and $\tau[d]$ denotes the sub-tree of τ rooted at position d.

We consider also a finite set of monadic [3] predicate symbols Pred; we will denote HB^*, the complete Herbrand base generated over Pred and T^*_Σ, *i.e.* the set of all ground atoms $p(\tau)$ with $p \in \mathsf{Pred}$ and $\tau \in T^*_\Sigma$.

For a term t or a first-order formula Ψ, $\mathit{Var}(t)$ and $\mathit{Var}(\Psi)$ will denote the set of all free variables respectively in t and in Ψ. Assuming that in the formula Ψ, two different quantifications always address two different variables, we denote $\mathit{Var}_\exists(\Psi)$ and $\mathit{Var}_\forall(\Psi)$ the set of all variables that are respectively existentially and universally quantified in Ψ.

2.1 Definitions

A *generalized clause* (or simply, a clause in the rest of the paper) is a first-order formula that generalizes Horn clauses by allowing universal quantification on the variables appearing in the body part of the clause. More formally, a generalized clause is an implication

$$p(t) \leftarrow \Psi \qquad \text{with } \Psi ::= p'(t') \mid \Psi \wedge \Psi \mid \exists y \Psi \mid \forall y \Psi \mid true$$

where t and t' s are first-order terms and p, p' belong to Pred. We assume wlog that the free variables of the formula Φ must occur in the head of the clause, that is in the term t.

Definition 1. *A Horn μ-program (P, Ω) is given by a set P of generalized clauses and a mapping Ω from Pred to the set of natural numbers.*

For a predicate p, $\Omega(p)$ is the *priority* of p and $\max(\Omega(P))$ is the maximal priority in the range of Ω over the predicates of P. For a clause c, $\Omega(c)$ denotes the priority of the predicate symbol occurring in the head of c.

The semantics for a Horn μ-program is given by means of ground proof trees. A proof tree for a ground atom $p(\tau)$ is a (possibly) infinite and infinitely-branching tree rooted in a node labeled by $p(\tau)$. Moreover, for any node n in the proof tree, labeled by some ground atom $p'(\tau')$, there exists a clause $p'(t) \leftarrow \Psi$ and a substitution $\sigma : \mathit{Var}(t) \mapsto T^*_\Sigma$ such that:

[3] For a matter of simplicity, we shall consider only monadic predicate symbols. The notions presented in this section extend naturally to predicate symbols with arbitrary arity.

- $\sigma(p'(t)) = p'(\tau')$ and,
- the sons of the node n are labeled by elements from the set $Sons(n, p'(t) \leftarrow \Psi)$ in such a way that for all elements l in $Sons(n, p'(t) \leftarrow \Psi)$, there is exactly one son of n labeled by l.

This set $Sons(n, p'(t) \leftarrow \Psi)$ is a set of ground atoms and is a minimal model over the universe HB^* of the formula Ψ under the substitution σ, *i.e.* $Sons(n, p'(t) \leftarrow \Psi), \sigma \models \Psi$.
Note that leaves in a proof tree correspond necessarily to the case where the clause used to compute $Sons(n, p'(t) \leftarrow \Psi)$ is a fact, *i.e.* $\Psi = true$ In this case, $Sons(n, p'(t) \leftarrow \Psi)$ is empty.
For an infinite path π in a proof tree, we denote *Inf*(π) the set of all priorities that occur infinitely often along the path π. We say that a proof tree accepts the atom $p(\tau)$ if it is rooted in a node labeled by $p(\tau)$ and for all paths π starting from the root, the maximal element of *Inf*(π) is even. A Horn μ-program (P, Ω) accepts a ground atom $p(\tau)$ if there exists a proof tree which accepts the atom $p(\tau)$.
We denote $[\![(P, \Omega)]\!]$ (resp. $[\![(P, \Omega), p]\!]$) the set of all ground atoms (resp. of all ground atoms for the predicate symbol p) accepted by the Horn μ-program (P, Ω).

2.2 The Alternation-Free Fragment of the Horn μ-Calculus

We give here a syntactic restriction for Horn μ-calculus as we presented it above. This restriction limits the dependency on predicates according to their respective priorities.
We say that a predicate p depends on a predicate q if there exists a clause $p(t) \leftarrow \Psi$ such that either q occurs in Ψ or some predicate r occurs in Ψ and r depends on q.

Definition 2 (Alternation-free Horn μ-calculus). *A Horn μ-program is said to be* alternation-free *if for any predicates p, q, if p depends on q then the priority of q is smaller or equal to the priority of p* i.e. $\Omega(q) \leq \Omega(p)$

Note that "classical" logic programs with respect to their usual least fix-point semantics as well as to their reactive greatest fix-point semantics fall into this alternation-free fragment: for a logic program P, the least fix-point semantics of P is exactly the denotation of the alternation-free Horn μ-program (P, Ω_l) where $\Omega_l(p) = 1$ for any predicate p in P whereas the greatest semantics is the denotation of (P, Ω_g) where $\Omega_g(p) = 0$ for any p in P.

We give for the alternation-free Horn μ-calculus a semantics in terms of fix-points. This latter is however equivalent to the ground proof tree semantics. For simplicity, we will assume from now on that the range of the function Ω is an interval over $\mathbb{N}$ of the form $[0 \ldots \max(\Omega(P))]$.

Let $T^i_{(P,\Omega)} : \mathsf{HB}^* \mapsto \mathsf{HB}^*$ be an operator defined for any integer i in the range of $\Omega(P)$ as follows:

$$T^i_{(P,\Omega)}(S) = (S \setminus \mathsf{HB}^*_i) \cup \left\{ \sigma(p(t)) \;\middle|\; \begin{array}{l} \text{there exists } p(t) \leftarrow \Psi \text{ in } P \text{ such that } \Omega(p) = i \\ \text{there exists a substitution } \sigma : \mathit{Var}(t) \mapsto T^*_\Sigma \\ \text{such that } S, \sigma \models \Psi \end{array} \right\}$$

where HB_i^* the set of all ground atoms defined for predicates p with priority i.
Let us consider a set A such that $A \cap \mathsf{HB}_i^* = \varnothing$. Now, it is easy to see that $L_A = (\{A \cup S \mid S \subseteq \mathsf{HB}_i^*\}, \subseteq)$ is a complete lattice and that $T^i_{(P,\Omega)}$ is monotonic over this lattice. So, by Knaster-Tarski'theorem, $T^i_{(P,\Omega)}$ admits a least and a greatest fix-points over this lattice respectively denoted $\mathit{lfp}_A(T^i_{(P,\Omega)})$ and $\mathit{gfp}_A(T^i_{(P,\Omega)})$.
In order to define the fix-point semantics for some alternation-free Horn μ-program (P, Ω), we introduce a family $(F^i_{(P,\Omega)})$ of sets of ground atoms for each i in $\Omega(P)$ as:

$$F^0_{(P,\Omega)} = \mathit{gfp}_{\varnothing}(T^0_{(P,\Omega)})$$

$$F^i_{(P,\Omega)} = \mathit{lfp}_{F^{i-1}_{(P,\Omega)}}(T^i_{(P,\Omega)}) \qquad \text{if } i \text{ is odd}$$

$$F^i_{(P,\Omega)} = \mathit{gfp}_{F^{i-1}_{(P,\Omega)}}(T^i_{(P,\Omega)}) \qquad \text{if } i \text{ is even}$$

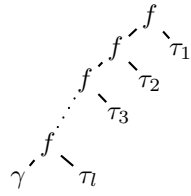
The fix-point semantics of (P, Ω) is simply the set $F^n_{(P,\Omega)}$ where n is $\max(\Omega(P))$. As claimed earlier, the fix-point and the proof tree semantics coincide as stated in the following theorem:

Theorem 1. *Let (P, Ω) be an alternation-free Horn μ-program with $\Omega(P) = [1 \ldots n]$. Then $[\![(P, \Omega)]\!] = F^n_{(P,\Omega)}$.*

Example 1. Let us consider (P, Ω), the alternation-free Horn μ-program given by

$$P = \left\{ \begin{array}{llll} p_0(f(x,y)) \leftarrow p_0(x), p_0(y) & p_0(a) & p_0(b) \\ p_1(f(x,y)) \leftarrow p_1(x), p_0(y) & p_1(a) & p_1(c) \\ p_2(x) \leftarrow \exists y\, p_1(f(x,y)) \wedge p_0(x) & & \end{array} \right\} \text{ and for all } i,\ \Omega(p_i) = i.$$

The set $F^0_{(P,\Omega)}$ is exactly the denotation of the predicate p_0, that is the set of all ground atoms $p_0(\tau)$ where τ is a finite or an infinite tree built over the binary function symbol f and the two constants a and b.
The set $F^1_{(P,\Omega)}$ contains $F^0_{(P,\Omega)}$ and the denotation of the predicate p_1. The denotation of this latter is the set of all ground atoms $p_1(\tau')$; these trees τ' consist of a finite left backbone of f symbols and terminated either with a or with c. Each of those f symbols from the backbone has as right son a finite or infinite tree built over f, a and b. The trees τ' can be depicted as



where γ is either a or c and the τ_i's are finite or infinite trees built over f, a and b.

Finally, the set $F^2_{(P,\Omega)}$ is the denotation of (P, Ω); it contains the set $F^1_{(P,\Omega)}$ and the denotation of p_2, that is the set of all ground atoms $p_2(\tau'')$, where τ'' is a finite left backbone of f terminated with the constant a and each right son of the f symbols from the backbone is a finite or infinite tree built over f, a and b. Hence, a tree τ'' is similar to the tree τ' depicted above except that for τ'', γ is necessarily the constant a.

3 Horn μ-Calculus for Uniform Programs

Uniform logic programs aimed to express the set-based analysis in terms of logic programs. They are the basis of the results from [1313] where it is shown that the Horn μ-calculus restricted to uniform programs is decidable. In this section, we investigate the case of the alternation-free Horn μ-calculus restricted to uniform programs that are extended with both existential and universal quantification in the body of clauses.

Definition 3. *A* uniform Horn μ-program *is given as a pair* (P, Ω) *where* Ω *is a priority function and* P *is a set of clauses (defined over a set* Pred *of monadic predicate symbols) of the following forms:*

$$p(f(x_1, \ldots, x_m)) \leftarrow p_1(x_1), \ldots, p_m(x_m)$$

$$p(x) \leftarrow \Psi$$

where: - $\Psi ::= p'(t') \mid \Psi \wedge \Psi \mid \exists y \Psi \mid \forall y \Psi \mid true$
- Ψ *contains at most* x *as a free variable,* i.e. $Var(\Psi) \subseteq \{x\}$.

The first kind of clause is called *automaton clause* as a reference the transition rule in a classical tree automaton. The second kind is called *projection clause*. From now on, the expression *uniform program* relates to a program of the form given in Definition 3.

It should be noticed that the alternation-free Horn μ-program given in Example 1 is uniform.

We have already mentioned the regularity aspect of the Horn μ-calculus restricted to uniform programs. Actually, this calculus can be viewed as a syntactic extension for the definition of Rabin tree automata. It turns out that Rabin tree automata, or to be more precise *parity tree automata*, correspond to uniform programs with clauses of the form $p(f(x_1, \ldots, x_m)) \leftarrow p_1(x_1), \ldots, p_n(x_n)$. Additionally, when uniform clauses of the form $p(x) \leftarrow q(x), r(x)$ are considered then those programs correspond to *alternating parity tree automata* [13].

As we have said before, we consider only programs based on a restriction of the Horn μ-calculus, namely the alternation-free fragment. Therefore, one may wonder whether there is an existing class of tree automata for which the alternation-free fragment of the Horn μ-calculus restricted to uniform programs is a generalization.

In [12] Muller and *al.* have introduced the notion of *weak-alternating tree automata*. They are based on a Büchi acceptance condition, that is on a set F such that a run is accepted iff for every path, the set of states occurring infinitely often intersects F. The main feature of weak-alternating tree automata are the following requirements:

- there exists a partition $(Q_0, \ldots, Q_n)$ of the set of states Q such that for all i, either $Q_i \subseteq F$ (Q_i is said to be *accepting*) or $Q_i \cap F = \varnothing$ (Q_i is said to be *rejecting*).
- the partition $(Q_0, \ldots, Q_n)$ is equipped with a partial ordering relation $\leq$,
- and, rephrased in our "uniform program" setting, for every transition rule in the automaton $p(f(x_1, \ldots, x_m)) \leftarrow p_1(x_1), \ldots, p_m(x_m)$ or $p(x) \leftarrow p_1(x), \ldots, p_m(x)$, for every p_i, if p belongs to Q_j and p_i belongs to Q_k, then $Q_k \leq Q_j$.

The encoding of the weak-alternating tree automata into the alternating-free fragment of the Horn μ-calculus is straightforward: it is sufficient to define a priority function Ω compatible with the ordering $\leq$ and such that $\Omega(q)$ is even if q belongs to an accepting set Q_i and $\Omega(q)$ is odd otherwise.

The main result of this paper can be stated as:

Theorem 2. *Let (P, Ω) be a uniform and alternation-free Horn μ-program. Deciding whether $[\![(P, \Omega), p]\!]$ is empty is* DEXPTIME*-complete.*

The DEXPTIME-hardness follows from [15]. To prove the completeness we develop a constructive approach: we consider in the next section a class of tree automata for finite and infinite trees, called monotonous tree automata. The main ingredient of these automata is a family of finite and complete lattices. States are defined component-wise as the product of elements of these lattices. Orderings for lattices are lifted to states and then, to runs. The acceptance condition is then expressed in terms of those orderings overs runs. Later on, we will give an algorithm that builds from a uniform and alternation-free Horn μ-program (P, Ω) a monotonous tree automaton $\mathcal{A}$. The basic idea is simply that the semantics of (P, Ω) coincide with the language accepted by $\mathcal{A}$.

As uniform and alternation-free Horn μ-programs extend weak-alternating tree automata, our method provides a new technique to check emptiness for those automata. Furthermore, emptiness problem for weak-alternating tree automata being DEXPTIME-hard, our method is theoretically optimal.

4 From Alternation-Free Horn μ-Programs to Tree Automata

4.1 Monotonous Tree Automata

We consider now a special kind of tree automata running on both finite and infinite trees.

Definition 4. *A* monotonous tree automaton *$\mathcal{A}$ is a triple $(\Sigma, (\mathcal{O}_i)_{i\in\{0,\ldots,k\}}, \Delta)$ where Σ is a finite signature and $(\mathcal{O}_i)_{i\in\{0,\ldots,k\}}$ is a family of complete finite lattices,* i.e. *each of the $\mathcal{O}_i = (S_i, \preceq^i)$ is a complete lattice over a finite set S_i. A state in $\mathcal{A}$ is a k-tuple $(s_0, \ldots, s_k)$ such each of the s_i's belongs to S_i. We denote Q the set of all states in $\mathcal{A}$. We assume each of the orderings $\preceq^i$ to be lifted in a component-wise way on states,* i.e. *$q \preceq^i q'$ iff for s_i, s'_i the i^{th} components of respectively q and q', $s_i \preceq^i s'_i$ holds. Δ is a set of transition rules $f(q_1, \ldots, q_n) \rightarrow q$ with $q_1, \ldots, q_n, q$ in Q and f in Σ, and moreover,*

- *Δ is deterministic: there exists at most one transition rule $f(q_1, \ldots, q_n) \rightarrow q$ for any $f, q_1, \ldots, q_n$.*
- *Δ is complete: there exists at least one transition rule $f(q_1, \ldots, q_n) \rightarrow q$ for any $f, q_1, \ldots, q_n$.*
- *the rules from the set Δ have the* monotonicity property*: for any two transition rules $f(q_1, \ldots, q_m) \rightarrow q$ and $f(q'_1, \ldots, q'_m) \rightarrow q'$, for any integer i in $\{0, \ldots, k\}$, if for all $j \leq i$ and for all $l \in 1..m$, $q_l \preceq^j q'_l$, then $q \preceq^i q'$.*

The *rank* of the automaton $\mathcal{A}$ given above is denoted *rank*$(\mathcal{A})$ and is equal to k; we denote $\mathcal{F}_{\preceq}$ the family of orderings $(\preceq^i)_{i\in\{o,\ldots,k\}}$.

We define for monotonous tree automata a notion of run that actually coincide with the classical one: a *run* $\mathsf{r} : dom(\tau) \mapsto Q$ for a tree τ in an automaton $\mathcal{A}$ is a mapping from the tree domain of τ to the set of states of the automaton. Moreover, this mapping r satisfies for any position d in $dom(\tau)$, labeled with a function symbol f (for arity m) and having $d.1$,..,$d.m$ as child positions, that $f(\mathsf{r}(d.1),\ldots,\mathsf{r}(d.m)) \rightarrow \mathsf{r}(d)$ is a transition rule of $\mathcal{A}$.

Note that since a monotonous tree automaton has to be (ascending) deterministic and complete, any finite tree τ admits a unique run in this automaton. Of course, this property is not true for infinite trees.

The acceptance condition is based on the family of orderings $\mathcal{F}_{\preceq} = (\preceq^0,\ldots,\preceq^k)$. We first extend each of those orderings $\preceq^i$ over runs as follows: for two runs r and r' (for a tree τ), $\mathsf{r} \preceq^i \mathsf{r}$' iff for any position d in the tree domain of τ, $\mathsf{r}(d) \preceq^i \mathsf{r}'(d)$.
We say that the set of all 0-accepting runs for a tree τ is the set of all runs for τ. Recursively, for $0 \leq i \leq k$, we say that r is a $(i+1)$-accepting run iff r is a minimal (resp. maximal) i-accepting run in the sense of $\preceq^i$ if i is odd (resp. even).

Definition 5 (Acceptance condition). *A run* r *for the tree* τ *in the automaton* $\mathcal{A}$ *is said to be* accepting *if* r *is a* $(k+1)$*-accepting run.*

Theorem 3. *For all trees* τ*, there exists a unique accepting run for* τ *in* $\mathcal{A}$*.*

Proof. We consider for any tree τ *and for* i *in* $\{0,\ldots,k+1\}$ *the set* $\mathcal{R}_i^{\tau,\mathcal{A}}$ *of* i*-accepting runs for* τ *in* $\mathcal{A}$*. We show that* $\mathcal{R}_0^{\tau,\mathcal{A}}$ *is not empty and that for* i *in* $\{0,\ldots,k\}$*,* $\mathcal{R}_i^{\tau,\mathcal{A}}$ *admits some minimal and maximal elements in the sense of* $\preceq^i$*. Thus, for* i *in* $\{0,\ldots,k+1\}$*, none of the* $\mathcal{R}_i^{\tau,\mathcal{A}}$*'s is empty. Therefore, any tree* τ *has at least one accepting run in* $\mathcal{A}$*. Finally, using the definition of states and of* i*-acceptance, it is easy to see that* $\mathcal{R}_{k+1}^{\tau,\mathcal{A}}$ *can be at most a singleton. See [17] for the details.*

From now on, we denote $\mathsf{r}_\tau^{\mathcal{A}}$ the unique accepting run for the tree τ in $\mathcal{A}$. A state q is *reachable* in the automaton $\mathcal{A}$ if there exists a tree τ such that for the unique accepting run $\mathsf{r}_\tau^{\mathcal{A}}$, $\mathsf{r}_\tau^{\mathcal{A}}(\epsilon) = q$. For a state q, we define $\mathcal{L}(\mathcal{A},q)$ the language accepted by the automaton $\mathcal{A}$ in this state q as the set of trees $\{\tau \in T_\Sigma^* \mid \mathsf{r}_\tau^{\mathcal{A}}(\epsilon) = q\}$.

Theorem 4. *The language* $\mathcal{L}(\mathcal{A},q)$ *is a regular set of trees.*

Proof. By reduction to SkS *(*k *being the maximal arity in* Σ*): for the language* $\mathcal{L}(\mathcal{A},q)$*, we construct a* SkS*-formula* $\psi_q^{\mathcal{A}}$ *such that the full k-ary tree containing a tree* τ *is a model of* $\psi_q^{\mathcal{A}}$ *iff* $\tau \in \mathcal{L}(\mathcal{A},q)$*. See [17] for the detailed proof.*

Note that the constructive proof for showing the regularity of $\mathcal{L}(\mathcal{A},q)$ can easily be adapted to show that reachability for a state can be encoded into SkS as well. Unfortunately, this would provide a quite high-complexity algorithm whereas as stated in the next theorem, reachability can be tested efficiently.

For the automaton $\mathcal{A}$, we define the size of $\mathcal{A}$ as $(|\mathcal{O}_0| \times \ldots \times |\mathcal{O}_k|)^{c_\Sigma}$, where $|\mathcal{O}_i|$ is the cardinality of the lattice $\mathcal{O}_i$ and c_Σ is a constant depending only on the signature Σ.

Theorem 5.

(i) Reachability for a state q can be decided in polynomial time in the size of the automaton $\mathcal{A}$.
(ii) The emptiness of the language $\mathcal{L}(\mathcal{A}, q)$ can be decided in polynomial time in the size of $\mathcal{A}$.

Proof. See [17] for the proof of (i). For (ii), checking emptiness for $\mathcal{L}(\mathcal{A}, q)$ simply amounts to check whether the state q is reachable. This can be achieved in polynomial time due to (i).

To conclude this section, let us say a few words about the expressiveness of monotonous tree automata. We have shown in Theorem 4 that the accepted language (for a fixed final state, and so, for a finite set of final states) is a Rabin regular tree language. On one hand, we will see in the next section how monotonous tree automata can be used to accept the same language as the ones defined by a uniform and alternation-free Horn μ-program. On the other hand, we have already said that these programs extend weak-alternating tree automata; furthermore, it is known that weak-alternating tree automata accept exactly languages that can be defined by Büchi tree automata whose complement is also a Büchi tree automaton (often denoted as Büchi $\cap$ co-Büchi). Therefore, monotonous tree automata are at least as expressive as Büchi $\cap$ co-Büchi. It is not yet known whether this inclusion is strict or not.
When one considers only signatures restricted to unary function symbols, then tree automata correspond to word automata. In this case, Theorem 4 claims that monotonous automata accept languages which are regular sets of words, *i.e.* ω-regular languages. On the other hand, it is know that weak-alternating word automata accept also exactly the ω-regular languages. Hence, when restricted to words (that is, unary function symbols), monotonous tree automata accept exactly the regular languages.
One should also notice that the class of monotonous tree automata is closed under complementation. For an automaton $\mathcal{A}$ whose states are in Q, let us consider $F \subseteq Q$ a set of final states. We define the language $\mathcal{L}(\mathcal{A}, F)$ as $\bigcup_{q \in F} \mathcal{L}(\mathcal{A}, q)$. We know already that this language is a regular set of trees. Due to the uniqueness of the accepting run for any tree τ, the complement of this language, $\overline{\mathcal{L}(\mathcal{A}, F)}$ is simply $\mathcal{L}(\mathcal{A}, Q \setminus F)$. Hence, given a monotonous tree automaton $\mathcal{A}$ together with a set of final states F, one can construct an automaton recognizing the complemented language in linear time.

4.2 Fix-Point Operators for Automata

In this section, we present a construction that given a uniform and alternation-free Horn μ-program computes a monotonous tree automaton such that the language recognized by the automaton coincides with the semantics of the program. The construction is based on an instantiation (depending on the program (P, Ω)) of the definition we gave for monotonous tree automata in the previous section: to be a little bit more explicit, states of these automata will be seen as sets of predicate symbols occurring in (P, Ω) and the family of orderings defining the acceptance condition will be expressed in terms of set inclusions taking the priority of predicates into account.

Let us consider (P, Ω), a uniform and alternation-free Horn μ-program for which the set of predicates is Pred. Pred_i is the set of all predicates in Pred with priority i; $\mathsf{Pred}_i = \{p \in \mathsf{Pred} \mid \Omega(p) = i\}$. We instantiate the notion of automata given in the previous section. This yields a finite family of automata, denoted $\mathcal{F}_{(P,\Omega)}$, satisfying:

- Σ is the set of function symbols over which P is defined,
- the family of finite and complete lattices is given by $\mathcal{O}_{i\in\{0,\ldots,\max(\Omega(P))\}}$, where for each i, $\mathcal{O}_i = (\wp(\mathsf{Pred}_i), \subseteq)$. $\wp(\mathsf{Pred}_i)$ denotes the set of all subsets of Pred_i. A state in these automata is a tuple of sets of predicate symbols such that the i^{th} component is a set that contains only predicate symbols of priority i. The ordering of the lattice $\mathcal{O}_i$ is simply the inclusion relation.

For convenience, we will regard a state simply as a unique set of predicate symbols. Since components of a tuple representing a state are pair-wise disjoint, the set of all predicate symbols occurring in a tuple corresponds to a unique state and vice-versa. This view simply imposes that for the family of orderings $(\preceq^0, \ldots, \preceq^n)$, each ordering $\preceq^i$ as to be defined as: $q \preceq^i q'$ iff $q \cap \mathsf{Pred}_i \subseteq q' \cap \mathsf{Pred}_i$ for any states q, q' viewed as sets of predicates.

The construction is defined for this instantiation by means of fix-point operators tranforming an automaton from $\mathcal{F}_{(P,\Omega)}$ into another one. These operators are defined for each clause: the definition of an operator depends on the priority of the predicate occurring in the head and the nature ("automaton"/"projection") of its clause.
Let us first sketch the basic idea of our approach: the semantics for a Horn μ-program (P, Ω) associates with each of its predicates a set of trees. This can be viewed the other way round: the semantics can be expressed as a unique mapping $\xi_{(P,\Omega)}$ from the set of trees to sets of predicates as a kind of characteristic function: for instance, for some tree τ, $\xi_{(P,\Omega)}(\tau)$ could be $\{p, q\}$ which stands for "both $p(\tau)$ and $q(\tau)$ belong to $[\![(P, \Omega)]\!]$ and for no other predicate r, $r(\tau)$ belongs to $[\![(P, \Omega)]\!]$".

Note that the mapping $\xi_{(P,\Omega)}$ defines also for a fixed tree τ a unique mapping from the tree domain of τ to sets of predicate symbols simply by associating to each position d in τ the value of $\xi_{(P,\Omega)}(\tau[d])$ (recall that $\tau[d]$ denotes the subtree of τ at position d). Therefore, $\xi_{(P,\Omega)}$ for a fixed tree τ is of the same kind as a run for the tree τ in a monotonous tree automaton from $\mathcal{F}_{(P,\Omega)}$. What we show here is that for $\xi_{(P,\Omega)}$, there exists a monotonous tree automaton $\mathcal{A}_{(P,\Omega)}$ such that for any tree τ, $\xi_{(P,\Omega)}$ over the tree domain of τ is exactly the accepting run for τ in $\mathcal{A}_{(P,\Omega)}$. Furthermore, we give an algorithm that computes this automaton $\mathcal{A}_{(P,\Omega)}$.

We have described so far the shape of the tree automata we have to consider (namely the family $\mathcal{F}_{(P,\Omega)}$) and gave briefly the intuition of what the construction should yield. We are now going to explicitly formalize this construction. Roughly speaking, this construction is an alternating fix-point computation that mimics somehow the fix-points semantics for alternation-free Horn μ-programs given in Section 2.2.
To make the link clearer, we need to rephrase the definition for the operators $T^i_{(P,\Omega)}$ in terms of contributions of each clause that is taken into account by this operator. For $T^i_{(P,\Omega)}$, these clauses will be the ones for which the predicate in their head has priority i. Let us denote C^i the set of such clauses. Hence, for any set of atoms S, one have

$$T^i_{(P,\Omega)}(S) = \bigcup_{c \in C^i} T^i_{(c,\Omega)}(S)$$

where $T^i_{(c,\Omega)}$ is defined as $T^i_{(P,\Omega)}$ for a program having c as a unique clause.

Our construction will mimic this rephrased definition: we will define for each priority i, an operator $T^i_{\mathcal{F}}$ over tree automata from $\mathcal{F}_{(P,\Omega)}$ as the "union" of individual operators $T_{\mathcal{F},c}$ defined for all clauses with an head of priority i. Thus, as fix-point operators introduced for the semantics transform a set of atoms into another set of atoms, the operators $T_{\mathcal{F},c}$ and $T^i_{\mathcal{F}}$ will transform a tree automaton from $\mathcal{F}_{(P,\Omega)}$ into another tree automaton.

We start with defining the "union" operator $\sqcup$ for monotonous tree automata,

Definition 6. *For $\mathcal{A}, \mathcal{A}'$ in $\mathcal{F}_{(P,\Omega)}$ having respectively $\Delta_{\mathcal{A}}$ and $\Delta_{\mathcal{A}'}$ as sets of transition rules, the monotonous tree automaton $\mathcal{A} \sqcup \mathcal{A}'$ has, as transition rules, the set*

$$\{\, lhs \to q \cup q' \mid lhs \to q \text{ in } \Delta_{\mathcal{A}},\ lhs \to q' \text{ in } \Delta_{\mathcal{A}'} \,\}$$

For each clause c from P we will introduce an operator $T_{\mathcal{F},c} : \mathcal{F}_{(P,\Omega)} \mapsto \mathcal{F}_{(P,\Omega)}$ and we define $T^i_{\mathcal{F}}$, the tree automata operator as,

$$T^i_{\mathcal{F}}(\mathcal{A}) = \bigsqcup_{c \in C^i} T_{\mathcal{F},c}(\mathcal{A})$$

Now for clauses c from P, the operators $T_{\mathcal{F},c}$ are defined in a generic way by distinguishing clauses according the priority of the predicate in their head and according to their nature (automaton *vs.* projection).
Let us start with the case where c is an automaton clause.

Definition 7. *Let $\mathcal{A}$ be a tree automaton from $\mathcal{F}_{(P,\Omega)}$ with $\Delta_{\mathcal{A}}$ as set of transition rules. Let $c = p(f(x_1, \ldots, x_m)) \leftarrow p_1(x_1), \ldots, p_m(x_m)$ be an automaton clause from P. The operator $T_{\mathcal{F},c}$ is defined according to the parity of $\Omega(p)$ as*

- *if $\Omega(p)$ is odd: the tree automaton $T_{\mathcal{F},c}(\mathcal{A})$ has for transition rules the set*

$$\left\{ lhs \to q' \;\middle|\; \begin{array}{l} lhs \to q \in \Delta_{\mathcal{A}} \text{ and} \\ q' = \begin{cases} q \cup \{p\} & \text{if } lhs = f(q_1 \ldots, q_m) \text{ and for all } i : p_i \in q_i \\ q & \text{otherwise} \end{cases} \end{array} \right\}$$

- *if $\Omega(p)$ is even: the tree automaton $T_{\mathcal{F},c}(\mathcal{A})$ has for transition rules the set*

$$\left\{ lhs \to q' \;\middle|\; \begin{array}{l} lhs \to q \in \Delta_{\mathcal{A}} \text{ and} \\ q' = \begin{cases} q \setminus \{p\} & \text{if } lhs \neq f(q_1 \ldots, q_m) \text{ or exists } i : p_i \in q_i \\ q & \text{otherwise} \end{cases} \end{array} \right\}$$

Defining $T_{\mathcal{F},c}$ for projection clauses is more involved and requires some auxiliary notions. We first point out an algebraic view of monotonous tree automata. The second step consists of a formalization of the different processing of quantified and free variables in the formulas from the body of clauses we have to consider.

We consider the finite algebra $\mathsf{A}_{\mathcal{A}}$ defined by a monotonous tree automaton $\mathcal{A}$; the carrier of $\mathsf{A}_{\mathcal{A}}$ is the set of all states of $\mathcal{A}$ and function symbols are interpreted according to transition rules: the function symbol f is interpreted in the algebra $\mathsf{A}_{\mathcal{A}}$ by a function $f^{\mathsf{A}_{\mathcal{A}}}$ such that for any tuple of states $q_1, \ldots, q_m$, $f^{\mathsf{A}_{\mathcal{A}}}(q_1, \ldots, q_m) = q$ iff $f(q_1, \ldots, q_m) \to q$ is a transition rule in $\mathcal{A}$. Note that we use here the fact that $\mathcal{A}$ is ascending deterministic and complete.
This algebra $\mathsf{A}_{\mathcal{A}}$ can be extended to a unique structure $\mathsf{R}_{\mathcal{A}}$ simply by interpreting the monadic predicate p from the program as the set of all states that contain this predicate. Formally, the semantics of p in $\mathsf{R}_{\mathcal{A}}$ is $\{q \in Q \mid p \in q\}$.
The main idea to address projection clauses is to consider this finite structure $\mathsf{R}_{\mathcal{A}}$ for interpreting formulas in the body of clauses.

But this very simple and natural idea requires an additional technical point: for a clause $p(x) \leftarrow \Psi(x)$, we want to interpret the formula $\Psi(x)$ in the finite structure of states $\mathsf{R}_{\mathcal{A}}$. However, for correctness, we have to consider the variable x ranging over arbitrary states, whereas for the quantified variables that may occur in $\Psi(x)$, we must consider them as ranging only over reachable states. To model formally this requirement, we will introduce a new formula $\tilde{\Psi}$ for each formula Ψ as follows:

- if Ψ is an atom $p(t)$ then $\tilde{\Psi} = p(t)$.
- if $\Psi = \Psi_1 \wedge \Psi_2$ (resp. $\Psi = \Psi_1 \vee \Psi_2$) then $\tilde{\Psi} = \tilde{\Psi}_1 \wedge \tilde{\Psi}_2$ (resp. $\tilde{\Psi} = \tilde{\Psi}_1 \vee \tilde{\Psi}_2$).
- if $\Psi = \exists y\, \Psi'$ then $\tilde{\Psi} = \exists y\, p_\top(y) \wedge \tilde{\Psi}'$,
- if $\Psi = \forall y\, \Psi'$ then $\tilde{\Psi} = \forall y\, p_\top(y) \Rightarrow \tilde{\Psi}'$.

The predicate symbol $p_\top$ is a new predicate symbol. Intuitively, from the Herbrand semantics point of view, its semantics is the set of all trees. One can imagine this predicate defined as $\forall x\, p_\top(x)$ when Herbrand structures are considered. For any quantified variable y in the formula Ψ, the corresponding variable y in the formula $\tilde{\Psi}$ will be *guarded* by an atom $p_\top(y)$.

As we said above, due to the particular interpretation of $p_\top$ in Herbrand structure,

Remark 1. For any Herbrand structure $\mathsf{R}_{\mathsf{HB}^*}$ defining semantics for the predicate symbols and such that the semantics of $p_\top$ is the set of all trees, $\mathsf{R}_{\mathsf{HB}^*}, [x/\tau] \models \Psi(x)$ holds iff $\mathsf{R}_{\mathsf{HB}^*}, [x/\tau] \models \tilde{\Psi}(x)$ holds.

However, things are going to be different in the automaton structure $\mathsf{R}_{\mathcal{A}}$. In this structure $\mathsf{R}_{\mathcal{A}}$, we fix the semantics of the predicate $p_\top$ to the set of all reachable states in $\mathcal{A}$. Hence, this will ensure in a formal way that for the formula Ψ, one considers quantified variables instantiated only with reachable states, whereas the free variable of this formula may take arbitrary states for values.

Definition 8. *Let $\mathcal{A}$ be a tree automaton from $\mathcal{F}_{(P,\Omega)}$ with $\Delta_{\mathcal{A}}$ as set of transition rules. Let $c = p(x) \leftarrow \Psi(x)$ be a projection clause from P. The operator $T_{\mathcal{F},c}$ is defined according to the parity of $\Omega(p)$ as*

– *if $\Omega(p)$ is odd: the tree automaton $T_{\mathcal{F},c}(\mathcal{A})$ has for transition rules the set*

$$\left\{ lhs \to q' \;\middle|\; \begin{array}{l} lhs \to q \in \Delta_{\mathcal{A}} \text{ and} \\ q' = \begin{cases} q \cup \{p\} & \text{if } \mathsf{R}_{\mathcal{A}}, [x/q] \models \tilde{\Psi}(x) \\ q & \text{otherwise} \end{cases} \end{array} \right\}$$

– *if $\Omega(p)$ is even: the tree automaton $T_{\mathcal{F},c}(\mathcal{A})$ has for transition rules the set*

$$\left\{ lhs \to q' \;\middle|\; \begin{array}{l} lhs \to q \in \Delta_{\mathcal{A}} \text{ and} \\ q' = \begin{cases} q \setminus \{p\} & \text{if } \mathsf{R}_{\mathcal{A}}, [x/q] \not\models \tilde{\Psi}(x) \\ q & \text{otherwise} \end{cases} \end{array} \right\}$$

Before carrying on the presentation of our algorithm, it may be worth to clarify a point about the operators $T^i_{\mathcal{F}}$ and $T_{\mathcal{F},c}$. We have claim earlier that those operators are defined over $\mathcal{F}_{(P,\Omega)}$. When it is quite clear that, due to their respective definition, $T^i_{\mathcal{F}}$ and $T_{\mathcal{F},c}$ associates with an automaton in $\mathcal{F}_{(P,\Omega)}$ a tree automaton defined over the same signature and the same set of state which is both (ascending) deterministic and complete, the monotonicity property is less straightforward for the resulting automaton. However, this is obvious that if the monotonicity property holds for $\mathcal{A}$ and $\mathcal{A}'$, then it holds for $\mathcal{A} \sqcup \mathcal{A}'$. This is also true for any of the $T_{\mathcal{F},c}$ operators as claimed in the next proposition

Proposition 1. *Let $\mathcal{A}$ be an automaton from $\mathcal{F}_{(P,\Omega)}$, then both $T_{\mathcal{F},c}(\mathcal{A})$ and $T^i_{\mathcal{F}}(\mathcal{A})$ satisfy the monotonicity property.*

Proof. The detailed proof for $T_{\mathcal{F},c}$ can be found in [17]. Taking into account the definition for $T^i_{\mathcal{F}}$ in terms of $\sqcup$ yields the proof.

We have yet introduced all the material needed to describe the algorithm. As we have said earlier, this latter simply mimics the computation of the fix-point semantics for (P, Ω) in terms of tree automata.
Let us consider an automaton $\mathcal{A}^{\perp}$ that belongs $\mathcal{F}_{(P,\Omega)}$ and that we call *initial*. This initial automaton is the one from $\mathcal{F}_{(P,\Omega)}$ having the right-hand side of its transition rules equals to a particular state denoted $q_{\mathcal{F}_{\preceq}}$. This state $q_{\mathcal{F}_{\preceq}}$ is the state satisfying: for all predicate symbols p, $p \in q_{\mathcal{F}_{\preceq}}$ iff $\Omega(p)$ is even.

The family of automata $(\mathcal{A}^i_{(P,\Omega)})_{i \in \Omega(P)}$ is given by:

$$\mathcal{A}^0_{(P,\Omega)} = (T^0_{\mathcal{F}})^*(\mathcal{A}^{\perp}) \qquad \text{and} \qquad \mathcal{A}^i_{(P,\Omega)} = (T^i_{\mathcal{F}})^*(\mathcal{A}^{i-1}_{(P,\Omega)}) \quad \text{for } 0 < i$$

where $(T^i_{\mathcal{F}})^*(\mathcal{A})$ is the unique automaton $\mathcal{A}'$ for which there exists a least integer m such that $\mathcal{A}' = \underbrace{T^i_{\mathcal{F}} \circ \ldots \circ T^i_{\mathcal{F}}}_{m \text{ times}}(\mathcal{A})$ and $T^i_{\mathcal{F}}(\mathcal{A}') = \mathcal{A}'$.
The fact that such an integer m exists follows directly from the definitions of $T_{\mathcal{F},c}$ and $\sqcup$ and from the definition for the initial automaton $\mathcal{A}^{\perp}$.

The output of the algorithm is the tree automaton $\mathcal{A}^n_{(P,\Omega)}$ where $n = \max(\Omega(P))$. It should be noticed that the computation of $\mathcal{A}^n_{(P,\Omega)}$ is not the computation of a model

of (P, Ω) on a finite algebra (or, finite pre-interpretation): performing the computation of a structure (that is, of an interpretation of predicate symbols) over a finite algebra would imply that the considered algebra is fixed all along the computation whereas the structure evolves. Contrary to this latter, in our approach, the finite algebra we consider changes during the computation according to the automata whereas the structure is fixed for a given algebra.
The correctness of our approach, that is the equivalence between the uniform and alternation-free Horn μ-program (P, Ω) and the tree automaton $\mathcal{A}^n_{(P,\Omega)}$ can be phrased as

Theorem 6. *A ground atom $p(\tau)$ belongs to $[\![(P, \Omega)]\!]$ iff $p \in \mathsf{r}_\tau^{\mathcal{A}^n_{(P,\Omega)}}(\epsilon)$ for $n = \max(\Omega(P))$.*

Proof. See [17] for the proof.

Example 2. We illustrate our algorithm with the uniform and alternation-free Horn μ-program (P, Ω) given in Example 1. For conciseness, we identify a tree automaton with its set of transition rules and use a meta-representation for states occurring in the left-hand sides of transition rules. A state is represented as a pair $[X, Y]$ such that $X, Y \subseteq \mathsf{Pred}$ and $X \cap Y = \varnothing$. The pair $[X, Y]$ represents all states that contain X and that do not contain Y. For instance, $[\{p_0\}, \{p_1\}]$ represents the states $\{p_0\}$ and $\{p_0, p_2\}$. As an extension, the (meta-)transition rule $g([\{p_0\}, \{p_1\}]) \to q$ represents the two transition rules $g(\{p_0\}) \to q$ and $g(\{p_0, p_2\}) \to q$. Finally, we simply refer to a predicate symbol with its index, *i.e.* 2 stands for p_2.
We compute the family of automata $(\mathcal{A}^i_{(P,\Omega)})_{i\in\{0,1,2\}}$. The automaton $\mathcal{A}^2_{(P,\Omega)}$ is equivalent to the denotation of (P, Ω) as stated in Theorem 6.
The initial automaton $\mathcal{A}^\perp$ is given by

$$\left\{\begin{array}{cc} a \to \{0,2\} & b \to \{0,2\} \quad c \to \{0,2\} \\ g([\varnothing,\varnothing]) \to \{0,2\} & f([\varnothing,\varnothing],[\varnothing,\varnothing]) \to \{0,2\} \end{array}\right\}$$

For $c_1 = p_0(a)$ and $c_2 = p_0(b)$, $T_{\mathcal{F},c_1}(\mathcal{A}^\perp)$ and $T_{\mathcal{F},c_2}(\mathcal{A}^\perp)$ are equal respectively

to $\left\{\begin{array}{c} a \to \{0,2\} \\ b \to \{2\} \\ c \to \{2\} \\ g([\varnothing,\varnothing]) \to \{2\} \\ f([\varnothing,\varnothing],[\varnothing,\varnothing]) \to \{2\} \end{array}\right\}$ and to $\left\{\begin{array}{c} a \to \{2\} \\ b \to \{0,2\} \\ c \to \{2\} \\ g([\varnothing,\varnothing]) \to \{2\} \\ f([\varnothing,\varnothing],[\varnothing,\varnothing]) \to \{2\} \end{array}\right\}$.

And for $c_3 = p_0(f(x, y)) \leftarrow p_0(x), p_0(y)$,

$$T_{\mathcal{F},c_3}(\mathcal{A}^\perp) = \left\{\begin{array}{cccc} a \to \{2\} & b \to \{2\} & c \to \{2\} & g([\varnothing,\varnothing]) \to \{2\} \\ f([\{0\},\varnothing],[\{0\},\varnothing]) \to \{0,2\} & & f([\varnothing,\{0\}],[\{0\},\varnothing]) \to \{2\} & \\ f([\{0\},\varnothing],[\varnothing,\{0\}]) \to \{2\} & & f([\varnothing,\{0\}],[\varnothing,\{0\}]) \to \{2\} & \end{array}\right\}$$

So, for $T^0_{\mathcal{F}}(\mathcal{A}^\perp) = T_{\mathcal{F},c_1}(\mathcal{A}^\perp) \sqcup T_{\mathcal{F},c_2}(\mathcal{A}^\perp) \sqcup T_{\mathcal{F},c_3}(\mathcal{A}^\perp)$, we have

$$\left\{\begin{array}{cccc} a \to \{0,2\} & b \to \{0,2\} & c \to \{2\} & g([\varnothing,\varnothing]) \to \{2\} \\ f([\{0\},\varnothing],[\{0\},\varnothing]) \to \{0,2\} & & f([\varnothing,\{0\}],[\{0\},\varnothing]) \to \{2\} & \\ f([\{0\},\varnothing],[\varnothing,\{0\}]) \to \{2\} & & f([\varnothing,\{0\}],[\varnothing,\{0\}]) \to \{2\} & \end{array}\right\}$$

It is easy to see that $T^0_{\mathcal{F}}(T^0_{\mathcal{F}}(\mathcal{A}^{\perp})) = T^0_{\mathcal{F}}(\mathcal{A}^{\perp})$. Therefore, $\mathcal{A}^0_{(P,\Omega)} = T^0_{\mathcal{F}}(\mathcal{A}^{\perp})$. The next computed automaton is then $T^1_{\mathcal{F}}(\mathcal{A}^0_{(P,\Omega)})$, which is equal to

$$\left\{\begin{array}{cc} a \to \{0,1,2\} & b \to \{0,2\} \quad c \to \{1,2\} \quad g([\varnothing,\varnothing]) \to \{2\} \\ f([\{0,1\},\varnothing],[\{0\},\varnothing]) \to \{0,1,2\} & f([\{0\},\{1\}],[\{0\},\varnothing]) \to \{0,2\} \\ f([\{1\},\{0\}],[\{0\},\varnothing]) \to \{1,2\} & f([\varnothing,\{0,1\}],[\{0\},\varnothing]) \to \{2\} \\ f([\{0\},\varnothing],[\varnothing,\{0\}]) \to \{2\} & f([\varnothing,\{0\}],[\varnothing,\{0\}]) \to \{2\} \end{array}\right\}$$

Once again it is easy to see that $T^1_{\mathcal{F}}(T^1_{\mathcal{F}}(\mathcal{A}^0_{(P,\Omega)})) = T^1_{\mathcal{F}}(\mathcal{A}^0_{(P,\Omega)})$. So, $\mathcal{A}^1_{(P,\Omega)} = T^1_{\mathcal{F}}(\mathcal{A}^0_{(P,\Omega)})$. In this automaton, all the states in the right-hand side of the transition rules are reachable (and of course, only those ones): the states $\{0,1,2\}$, $\{0,2\}$, $\{1,2\}$ and $\{2\}$ are reachable since they label the root of the unique (and thus, accepting) run for respectively a, b, c and $g(a,a)$. Now, for computing the automaton $T^2_{\mathcal{F}}(\mathcal{A}^1_{(P,\Omega)})$, one has to check for each state q in the right-hand side of transition rules whether, in the structure induced by $\mathcal{A}^1_{(P,\Omega)}$, the valuation $[x/q]$ renders the formula $\exists y\,(y \in \top \wedge p_1(f(x,y))) \wedge p_0(x)$ true or not. This formula is falsified by any state q which doesn't contain $\{0\}$. Hence, by definition of $T^2_{\mathcal{F}}$, $T^2_{\mathcal{F}}(\mathcal{A}^1_{(P,\Omega)})$ will contain the following (meta-)transition rules

$$\left\{\begin{array}{cc} c \to \{1\} & g([\varnothing,\varnothing]) \to \varnothing \\ f([\{1\},\{0\}],[\{0\},\varnothing]) \to \{1\} & f([\varnothing,\{0,1\}],[\{0\},\varnothing]) \to \varnothing \\ f([\{0\},\varnothing],[\varnothing,\{0\}]) \to \varnothing & f([\varnothing,\{0\}],[\varnothing,\{0\}]) \to \varnothing \end{array}\right\}$$

Now for the other rules, let us start with $q = \{0,1,2\}$ (that is the rules for a and for $f([\{0,1\},\varnothing],[\{0\},\varnothing])$). It is possible to find a reachable state q' such that $[x/q, y/q'] \models p_1(f(x,y))$. For instance due to the rule $f(\{0,1,2\},\{0,1,2\}) \to \{0,1,2\}$ in $\mathcal{A}^1_{(P,\Omega)}$ whose right-hand side contains 1, the state $\{0,1,2\}$ is a proper choice for q'. So, $T^2_{\mathcal{F}}(\mathcal{A}^1_{(P,\Omega)})$ contains the unchanged rules

$$a \to \{0,1,2\} \qquad f([\{0,1\},\varnothing],[\{0\},\varnothing]) \to \{0,1,2\}$$

On the other hand, for the remaining rules with $\{0,2\}$ as right-hand side, one can check that for $q = \{0,2\}$, it is not possible to find a reachable state q' such that $[x/q, y/q'] \models p_1(f(x,y))$. Therefore, $T^2_{\mathcal{F}}(\mathcal{A}^1_{(P,\Omega)})$ contains the rules

$$b \to \{0\} \qquad f([\{0\},\{1\}],[\{0\},\varnothing]) \to \{0\}$$

It is easy to check that $T^2_{\mathcal{F}}(T^2_{\mathcal{F}}(\mathcal{A}^1_{(P,\Omega)})) = T^2_{\mathcal{F}}(\mathcal{A}^1_{(P,\Omega)})$. So, $\mathcal{A}^2_{(P,\Omega)} = T^2_{\mathcal{F}}(\mathcal{A}^1_{(P,\Omega)})$. Let us present the relevant part of the automaton $\mathcal{A}^2_{(P,\Omega)}$, that is restricted over states occurring in the right-hand side of transition rules.

$$\begin{array}{cccc} & a \to \{0,1,2\} & b \to \{0\} & c \to \{1\} \\ g(\varnothing) \to \varnothing & g(\{0\}) \to \varnothing & g(\{1\}) \to \varnothing & g(\{0,1,2\}) \to \varnothing \\ f(\varnothing,\varnothing) \to \varnothing & f(\varnothing,\{0\}) \to \varnothing & f(\varnothing,\{1\}) \to \varnothing & f(\varnothing,\{0,1,2\}) \to \varnothing \\ f(\{0\},\varnothing) \to \varnothing & f(\{0\},\{0\}) \to \{0\} & f(\{0\},\{1\}) \to \varnothing & f(\{0\},\{0,1,2\}) \to \{0\} \\ f(\{1\},\varnothing) \to \varnothing & f(\{1\},\{0\}) \to \{1\} & f(\{1\},\{1\}) \to \varnothing & f(\{1\},\{0,1,2\}) \to \{1\} \\ & f(\{0,1,2\},\varnothing) \to \varnothing & f(\{0,1,2\},\{0\}) \to \{0,1,2\} & \\ & f(\{0,1,2\},\{1\}) \to \varnothing & f(\{0,1,2\},\{0,1,2\}) \to \{0,1,2\} & \end{array}$$

One can notice for this automaton that if a tree τ contains one occurrence of the function symbol g, then any run r for τ will satisfy $\mathsf{r}(\epsilon) = \varnothing$. So, τ does not belong to the denotation of any of the predicates p_0, p_1 and p_2.
Now let us consider f^ω the infinite binary tree with all its nodes labeled by f. This tree f^ω admits three runs in $\mathcal{A}^2_{(P,\Omega)}$. The first run r_1 associates with each node the state $\varnothing$, the second run r_2 the state $\{0\}$ and the last run r_3 the state $\{0,1,2\}$. r_1 can not be the accepted run since r_2 and r_3 are greater than r_1 in the sense of $\preceq^0$. Finally, r_2 is the accepted run since it is smaller in the sense of $\preceq^1$ than r_3. So, the tree f^ω belongs to the denotation of the predicate p_0 but not to the denotation of p_1 and p_2.
Finally, let us consider the two trees τ_1 and τ_2: τ_1 is a finite left backbone of f terminated with the constant c and each f from the backbone has f^ω as right son. The tree τ_2 is similar to τ_1 except that the backbone ends up with the constant a. Thus, the accepted run r_{τ_1} for τ_1 associates with each node outside of the backbone the state $\{0\}$ and because of the two rules $c \to \{1\}$ and $f(\{1\},\{0\}) \to \{1\}$, with each node from the backbone the state $\{1\}$. Therefore, τ_1 belongs only to the denotation of p_1. The accepted run r_{τ_2} for τ_2 is similar to r_{τ_1} outside of the backbone and, due to the rules $a \to \{0,1,2\}$ and $f(\{0,1,2\},\{0\}) \to \{0,1,2\}$, associates the state $\{0,1,2\}$ with the root of τ_2. Therefore, the tree τ_2 belongs to the denotation of p_0, p_1 and p_2.

The complexity for the construction of $\mathcal{A}^n_{(P,\Omega)}$ can be roughly estimated: the size for each automaton in the family $\mathcal{F}_{(P,\Omega)}$ is single-exponential in the size of P. Moreover, the basic operations $T_{\mathcal{F},c}$ and $\sqcup$ can be performed in polynomial time in the size of the automaton. Thus, computing $T^i_{\mathcal{F}}$ can be achieved by a polynomial-time algorithm in the size of the automaton. For each step from $\mathcal{A}^i_{(P,\Omega)}$ to $\mathcal{A}^{i+1}_{(P,\Omega)}$, the operator $T^i_{\mathcal{F}}$ has to be iterated. However, taking into account the definition for the basic operations and the initial automaton, we can claim that the number of iterations is bounded by a single-exponential in the size of P. So, globally, the automaton $\mathcal{A}^n_{(P,\Omega)}$ can be computed with an algorithm running in single-exponential time in the size of P.

By Theorem 6, testing whether $[\![(P,\Omega),p]\!]$ is empty simply amounts to search for reachability in the automaton $\mathcal{A}^n_{(P,\Omega)}$ for a state containing p. Then, by combining the complexity for the construction of $\mathcal{A}^n_{(P,\Omega)}$ and Theorem 5, the result claimed earlier in Theorem 2 follows.

Acknowledgments The author thanks Viorica Sofronie-Stokkermans and Witold Charatonik for discussions and reading. The author is grateful to Sophie Tison for her careful reading of a preliminary version of this paper. Finally, the author is deeply grateful to Damian Niwiński who gave him a crucial idea for the emptiness test of monotonous tree automata.

References

1. W. Charatonik, D. McAllester, D. Niwiński, A. Podelski, and I. Walukiewicz. The Horn Mu-Calculus. In *Proceedings of the* 13^{th} *IEEE Symposium on Logic in Computer Science*, pages 58–69, 1998. 418, 418, 424

2. W. Charatonik, D. McAllester, and A. Podelski. The Greatest Fixed Point of the τ_P abstraction is Regular. Seminar on Applications of Tree Automata in Rewriting, Logic and Programming, oct 1997. 419
3. W. Charatonik, D. Niwiński, and A. Podelski. Model checking for uniforms programs. Draft, 1998. 418, 419, 420, 420, 424
4. W. Charatonik and A. Podelski. Co-definite Set Constraints. In *Proceedings of the* 9^{th} *International Conference on Rewriting Techniques and Applications*, LNCS, 1998. 419
5. P. Devienne, J-M. Talbot, and S. Tison. Solving Classes of Set Constraints with Tree Automata. In G. Smolka, editor, *Proceedings of the* 3^{rd} *International Conference on Principles and Practice of Constraint Programming*, LNCS 1330, pages 62–76, oct 1997. 419
6. P. Devienne, J-M. Talbot, and S. Tison. Co-definite Set Constraints with Membership Expressions. In Joxan Jaffar, editor, *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming*, pages 25–39. MIT-Press, jun 1998. 419, 419
7. E. A. Emerson and C. S. Jutla. Tree Automata, Mu-Calculus and Determinacy. In IEEE, editor, *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 368–377. IEEE Computer Society Press, October 1991. 419
8. T. Frühwirth, E. Shapiro, M.Y. Vardi, and E. Yardeni. Logic Programs as Types for Logic Programs. In *Proceedings of the* 6^{th}*IEEE Symposium on Logic in Computer Science*, pages 300–309, jun 1991. 419, 419
9. N. Heintze and J. Jaffar. A Decision Procedure for a Class of Herbrand Set Constraints. In *Proceedings of the* 5^{th} *IEEE Symposium on Logic in Computer Science*, pages 42–51, jun 1990. 419
10. N. Heintze and J. Jaffar. A Finite Presentation Theorem for Approximating Logic Programs. In *Proceedings of the* 17^{th}*ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 197–209, jan 1990. 419, 419
11. J. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987. 418
12. D. E. Muller, A. Saoudi, and P. E. Schupp. Weak Alternating Automata Give a Simple Explanation of Why Most Temporal and Dynamic Logics are Decidable in Exponential Time. In *Proceedingsof the Third IEEE Symposium on Logic in Computer Science*, pages 422–427. IEEE Computer Society, jul 1988. 424
13. D. E. Muller and P. E. Schupp. Alternating Automata on Infinite Trees. *Theoretical Computer Science*, 54(2–3):267–276, 1987. 424
14. M.O. Rabin. Decidability of Second-order Theories and Automata on Infinite Trees. In *Transactions of American Mathematical Society*, volume 141, pages 1–35, 1969. 419
15. H. Seidl. Haskell Overloading is DEXPTIME-complete. *Information Processing Letter*, 52:57–60, 1994. 425
16. J-M. Talbot. *Contraintes Ensemblistes Définies et Co-définies : Extensions et Applications*. PhD thesis, Université des Sciences et Technologies de Lille, jul 1998. 419
17. J-M Talbot. On the alternation-free horn mu-calculus. Technical report, Max-Planck-Institut für Informatik, 2000. Long version of LPAR'00. 421, 426, 426, 427, 431, 432
18. J-M. Talbot, P. Devienne, and S. Tison. Generalized Definite Set Constraints. *CONSTRAINTS - An International Journal*, 5(1):161–202, 2000. 419