Extensions of Attribute Grammars for Structured Document Queries*

Frank Neven†

Abstract

Document specification languages like for instance XML, model documents using extended context-free grammars. These differ from standard context-free grammars in that they allow arbitrary regular expressions on the right-hand side of productions. To query such documents, we introduce a new form of attribute grammars (extended AGs) that work directly over extended context-free grammars rather than over standard context-free grammars. Viewed as a query language, extended AGs are particularly relevant as they can take into account the inherent order of the children of a node in a document. We show that two key properties of standard attribute grammars carry over to extended AGs: efficiency of evaluation and decidability of well-definedness. We further characterize the expressiveness of extended AGs in terms of monadic second-order logic and establish the complexity of their non-emptiness and equivalence problem to be complete for EXPTIME. As an application we show that the Region Algebra expressions can be efficiently translated into extended AGs. This translation drastically improves the known upper bound on the complexity of the emptiness and equivalence test for Region Algebra expressions from non-elementary to EXPTIME.

*A preliminary version of this paper was presented at the 7th International International Workshop on Database Programming Languages, Kinloch Rannoch, Scotland, 1999.

†Research Assistant of the Fund for Scientific Research, Flanders. Limburgs Universitair Centrum, Universitaire Campus, Dept. WNI, Infolab, B-3590 Diepenbeek, Belgium. E-mail: frank.neven@luc.ac.be. Phone: +32-(0)11-26.82.31. Fax: +32-(0)11-26.82.99.

1 Introduction

Structured document databases can be seen as derivation trees of some grammar which functions as the "schema" of the database [1, 2, 5, 23, 24, 27, 38, 43]. Document specification languages like, e.g., XML [15], model documents using *extended* context-free grammars. Extended context-free grammars (ECFG) are context-free grammars (CFG) having regular expressions over grammar symbols on the right-hand side of productions. It is known that ECFGs generate the same class of string languages as CFGs. Hence, from a formal language point of view, ECFGs are nothing but shorthands for CFGs. However, when grammars are used to model documents, i.e., when also the derivation trees are taken into consideration, the difference between CFGs and ECFGs becomes apparent. Indeed, compare Figure 1 and Figure 2. They both model a list of poems, but the CFG needs the extra non-terminals PoemList, VerseList, WordList, and LetterList to allow for an arbitrary number of poems, verses, words, and letters. These non-terminals, however, have no meaning at the level of the logical specification of the document.

A crucial difference between derivation trees of CFGs and derivation trees of ECFGs is that the former are ranked while the latter are not. In other words, nodes in a derivation tree of an ECFG need not have a fixed maximal number of children. While ranked trees have been studied in depth [20, 45], unranked trees only recently received new attention in the context of SGML and XML. Based on work of Pair and Quere [39] and Takahashi [44], Murata defined a bottom-up automaton model for unranked trees [32]. This required describing transition functions for an arbitrary number of children. Murata's approach is the following: a node is assigned a state by checking the sequence of states assigned to its children for membership in a regular language. In this way, the "infinite" transition function is represented in a finite way. We will extend this idea to attribute grammars. Brüggemann-Klein, Murata and Wood initiated an extensive study of tree automata over unranked trees [11].

The classical formalism of *attribute grammars*, introduced by Knuth [30], has always been a prominent framework for expressing computations on derivation trees. Attribute grammars provide a mechanism for annotating the nodes of a tree with so-called "attributes", by means of so-called "semantic rules" which can work either bottom-up (for so-called "synthesized" attribute values) or top-down (for so-called "inherited" attribute values). This formalism is successfully applied in such diverse fields of computer sci-

ence as compiler construction and software engineering (for a survey, see [18]). In previous work, we approached attribute grammars from a different direction, we investigated them as a query language for derivation trees of CFGs [34, 37, 38].

Inspired by the above mentioned idea of representing transition functions for automata on unranked trees as regular string languages, we introduce extended attribute grammars (extended AGs) that work directly over ECFGs rather than over standard CFGs. The main difficulty in achieving this is that the right-hand sides of productions contain regular expressions that, in general, specify infinite string languages. This gives rise to two problems for the definition of extended AGs that are not present for standard AGs:

(i) in a production, there may be an unbounded number of grammar symbols for which attributes should be defined; and

(ii) the definition of an attribute should take into account that the number of attributes it depends on may be unbounded.

We resolve these problems in the following way. For (i), we only consider unambiguous regular expressions in the right-hand sides of productions.[1] This means that every child of a node derived by the production $p = X \to r$ corresponds to exactly one position in r. We then define attributes uniformly for every position in r and for the left-hand side of p. For (ii), we only allow a finite set D as the semantic domain of the attributes and we represent semantic rules as regular languages over D much in the same way tree automata over unranked trees are defined.

By carefully tailoring the semantics of inherited attributes, extended AGs can take into account the inherent order of the children of a node in a document. This makes extended AGs particularly relevant as a query language. Indeed, as argued by Suciu [43], achieving this capability is one of the major challenges when applying the techniques developed for semi-structured data [1] to XML-documents.

An important subclass of queries in the context of structured document databases, are the queries that select those subtrees in a document that satisfy a certain pattern [4, 3, 28, 29, 33, 40]. These are essentially unary queries: they map a document to a set of its nodes. Extended AGs are

[1] This is no loss of generality, as any regular language can be denoted by an unambiguous regular expression [9]. SGML is even more restrictive as it allows only *one*-unambiguous regular languages [10, 47].

$$
\begin{aligned}
&\mathrm{DB} \to \mathrm{PoemList}\\
&\mathrm{PoemList} \to \mathrm{Poem\ PoemList}\\
&\mathrm{PoemList} \to \mathrm{Poem}\\
&\mathrm{Poem} \to \mathrm{VerseList}\\
&\mathrm{VerseList} \to \mathrm{Verse\ VerseList}\\
&\mathrm{VerseList} \to \mathrm{Verse}\\
&\mathrm{Verse} \to \mathrm{WordList}\\
&\mathrm{WordList} \to \mathrm{Word\ WordList}\\
&\mathrm{WordList} \to \mathrm{Word}\\
&\mathrm{Word} \to \mathrm{LetterList}\\
&\mathrm{LetterList} \to \mathrm{Letter\ LetterList}\\
&\mathrm{LetterList} \to \mathrm{Letter}\\
&\mathrm{Letter} \to \mathrm{a} \mid \ldots \mid \mathrm{z}
\end{aligned}
$$

Figure 1: A CFG modeling a list of poems.

$$
\begin{aligned}
&\mathrm{DB} \to \mathrm{Poem}^{+}\\
&\mathrm{Poem} \to \mathrm{Verse}^{+}\\
&\mathrm{Verse} \to \mathrm{Word}^{+}\\
&\mathrm{Word} \to (\mathrm{a} + \cdots + \mathrm{z})^{+}
\end{aligned}
$$

Figure 2: An ECFG modeling a list of poems.

especially tailored to express such unary queries: the result of an extended AG consists of those nodes for which the value of a designated attribute equals 1.[2]

The contributions of this paper can be summarized as follows:

1. We introduce extended attribute grammars as a query language for structured document databases defined by ECFGs. Queries in this query language can be evaluated in time quadratic in the number of nodes of the tree. We show that non-circularity, the property that an attribute grammar is well-defined for every tree, is in EXPTIME. Interestingly, the naive reduction of the non-circularity problem of extended AGs to the same problem for standard AGs gives rise to a double exponential algorithm. We obtain an EXPTIME upper bound by reducing

[2]We always assume that D contains the values 0 and 1 (*false* and *true*).

the problem to the problem of deciding whether a tree-walking automaton (over unranked trees) cycles. We then show the latter problem to be complete for EXPTIME. The EXPTIME upper bound for the non-circularity test of extended AGs is also a lower bound since deciding non-circularity for standard attribute grammar is already known to be hard for EXPTIME [26].

2. We generalize our earlier results on standard attribute grammars [6, 38] by showing that extended AGs express precisely the unary queries definable in monadic second-order logic (MSO). The difficult case consists of showing that extended AGs can compute the MSO-equivalence type of the input tree [35]. The only complication, compared to the case of standard attribute grammars, arises from the fact that derivation trees are now unranked.

3. We obtain the exact complexity of some relevant optimization problems for extended AGs. Concretely, we establish the EXPTIME-completeness of the non-emptiness (given an extended AG, does there exist a tree of which a node is selected by this extended AG?) and of the equivalence problem of extended AGs. Interestingly, in obtaining this result and the previous complexity result, we make use of nondeterministic two-way automata with a pebble to succinctly describe regular string languages. The crucial property of those, is that they can be transformed into nondeterministic one-way automata with only exponential size increase, as opposed to the expected double exponential size increase. The latter is a result due to Globerman and Harel [22].

4. We show that Region Algebra expressions (introduced by Consens and Milo [14]) can be simulated by extended AGs. Stated as such, the result is hardly surprising, since the former essentially corresponds to a fragment of first-order logic over trees while the latter corresponds to full MSO. We, however, exhibit an *efficient* translation, which gives rise to a drastic improvement on the complexity of the equivalence problem of Region Algebra expressions. To be precise, Consens and Milo first translate each Region Algebra expression into an equivalent first-order logic formula on trees and then invoke the known algorithm testing decidability of such formulas. Unfortunately, the latter algorithm has non-elementary complexity. That is, the complexity of this algorithm

cannot be bounded by an elementary function (i.e., an iterated exponential $2^{\wedge}(2^{\wedge}\ldots(2^n))$ where n is the size of the input). This approach therefore conceals the real complexity of the equivalence test of Region Algebra expressions. Our efficient translation of Region Algebra expressions into extended AGs, however, gives an EXPTIME algorithm. The thus obtained upper bound more closely matches the already known coNP lower bound [14].

5. We define *relational* extended AGs. These are a generalization of relational BAGs studied in [34], which in turn are based upon of relational attribute grammars as introduced by Courcelle and Deransart [16]. Relational extended AGs can express queries in various ways. We consider two of them and show that they do not increase the expressiveness of the formalism.

This paper is further organized as follows. In Section 2, we recall some basic definitions. In Section 3, we give an example introducing the important ideas for the definition of extended AGs which are introduced in Section 4. In Section 5, we obtain the exact complexity of the non-circularity test for extended AGs. In Section 6, we characterize the expressiveness of extended AGs in terms of monadic second-order logic. In Section 7, we establish the exact complexity of the emptiness and equivalence problem of extended AGs. We then use this result to improve the complexity of the emptiness and equivalence problem of Region Algebra expressions in Section 8. Finally, in Section 9, we introduce relational extended AGs. We present some concluding remarks in Section 10.

2 Basic definitions

We start by introducing the necessary notions to define extended AGs. More concretely, we recall the definition of unambiguous regular expressions and define tree automata over unranked trees on which extended AGs are inspired.

In all of the following, let Σ be a finite alphabet. We denote the length of a string w by $|w|$ and its i-th letter by w_i.

2.1 Unambiguous regular expressions

As is customary, we denote by $L(r)$ the language defined by the regular expression r. Further, we denote by $\mathrm{Sym}(r)$ the set of Σ-symbols occurring in r. The *marking* $\tilde{r}$ of r is obtained by subscripting in r the first occurrence of a symbol of $\mathrm{Sym}(r)$ by 1, the second by 2, and so on. For example, $a_1(a_2+b_3^*)^*a_4$ is the marking of $a(a+b^*)^*a$. We let $|r|$ denote the number of occurrences of Σ-symbols in r, while $r(i)$ denotes the Σ-symbol at the i-th occurrence in r for each $i \in \{1, \ldots, |r|\}$. Let $\widetilde{\Sigma}$ be the alphabet obtained from Σ by subscripting every symbol by all natural numbers, i.e., $\widetilde{\Sigma} := \{a_i \mid a \in \Sigma, i \in \mathbb{N}\}$. If $w \in \widetilde{\Sigma}^*$ then $w^\#$ denotes the string obtained from w by dropping the subscripts.

In the definition of extended AGs we shall restrict ourselves to unambiguous regular expressions defined as follows:

Definition 2.1 A regular expression r over Σ is *unambiguous* if for all $v, w \in L(\tilde{r})$, $v^\# = w^\#$ implies $v = w$.

That is, a regular expression r is unambiguous if every string in $L(r)$ can be matched to r in only one way. For example, the regular expression $(a+b)^*$ is unambiguous while $(aa+a)^*$ is not. Indeed, it is easily checked that the string aa can be matched to $(aa+a)^*$ in two different ways.

The following proposition, obtained by Book et al. [9], says that the restriction to unambiguous regular expressions is no loss of generality.

Proposition 2.2 *For every regular language R there exists an unambiguous regular expression r such that $L(r) = R$.*

As usual, a *nondeterministic finite automaton* M (NFA) over Σ is a tuple $(S, \Sigma, \delta, I, F)$ where S is finite set of states, $\delta : S \times \Sigma \to 2^S$ is the transition function, $I \subseteq S$ is the set of initial states, and $F \subseteq S$ is the set of final states. We denote the canonical extension of the transition function to strings by δ^*. A string $w \in \Sigma^*$ is accepted by M if $\delta^*(s_0, w) \in F$ for an $s_0 \in I$. The language accepted by M, denoted by $L(M)$, is defined as the set of all strings accepted by M. The size of M is defined as $|S| + |\Sigma|$.

A *state assignment* ρ of M for a string $w \in \Sigma^*$ is a mapping from $\{1, \ldots, |w|\}$ to S. A state assignment ρ for w is *valid* if there exists an $s_0 \in I$ such that $\rho(1) \in \delta(s_0, w_1)$, $\rho(|w|) \in F$, and for $i = 1, \ldots, |w| - 1$, $\rho(i+1) \in \delta(\rho(i), w_{i+1})$. Clearly, w is accepted by M if and only if there exists a valid state assignment for w.

For every unambiguous regular expression r there exists an NFA M_r with the property that can be informally stated as follows: if $w \in L(r)$ then there exists only one path in M_r that accepts w. That is, M_r can accept w only in one manner. We introduce some more notation to define this automaton M_r.

If w is a string and r is an unambiguous regular expression with $w \in L(r)$, then $\tilde{w}_r$ denotes the unique string over $\widetilde{\Sigma}$ such that $\tilde{w}_r^{\#} = w$ and $\tilde{w}_r \in L(\tilde{r})$. For $i = 1, \ldots, |w|$, define $\mathrm{pos}_r(i, w)$ as the subscript of the i-th letter in $\tilde{w}_r$. Intuitively, $\mathrm{pos}_r(i, w)$ indicates the position in r matching the i-th letter of w. For example, if $r = a(b + a)^*$ and $w = abba$, then $\tilde{r} = a_1(b_2 + a_3)^*$ and $\tilde{w}_r = a_1 b_2 b_2 a_3$. Hence,

$$\mathrm{pos}_r(1, w) = 1, \quad \mathrm{pos}_r(2, w) = 2, \quad \mathrm{pos}_r(3, w) = 2, \text{ and } \mathrm{pos}_r(4, w) = 3.$$

The following lemma is obtained by Book et al. [9].

Lemma 2.3 *For every unambiguous regular expression r there exists an NFA M_r over the states $\{0, \ldots, |r|\}$ with start state 0 such that*

1. *$L(r) = L(M_r)$;*
2. *for every string $w \in L(r)$ there exists only one valid state assignment ρ_w of M_r for w; and*
3. *for $i = 1, \ldots, n$, $\rho_w(i) = \mathrm{pos}_r(i, w)$.*

Moreover, M_r can be constructed in time polynomial in the size of r.

Proviso 2.4 In the remaining, when we say *regular expression*, we always mean *unambiguous* regular expression.

2.2 Trees

In this paper we only consider trees where the children of a node are ordered and carry a label from some finite alphabet Σ. We refer to such trees as Σ-trees. We introduce some terminology.

Trees will be denoted by the boldface characters $\mathbf{t}$, $\mathbf{s}$, $\mathbf{s}_1$, ..., while nodes of trees are denoted by $\mathbf{n}$, $\mathbf{m}$, $\mathbf{n}_1$, ... We use the following convention: if $\mathbf{n}$ is a node of a tree $\mathbf{t}$, then $\mathbf{n}i$ denotes the i-th child of $\mathbf{n}$. We denote the set of nodes of $\mathbf{t}$ by $\mathrm{Nodes}(\mathbf{t})$ and the root of $\mathbf{t}$ by $\mathrm{root}(\mathbf{t})$. Further, the *arity* of a

node $\mathbf{n}$ in a tree, denoted by $arity(\mathbf{n})$, is the number of children of $\mathbf{n}$. We say that a tree $\mathbf{t}$ has *rank* m, for $m \in \mathbb{N}$, if $arity(\mathbf{n}) \leq m$ for every $\mathbf{n} \in \text{Nodes}(\mathbf{t})$. The subtree of $\mathbf{t}$ rooted at $\mathbf{n}$ is denoted by $\mathbf{t_n}$; the *envelope* of $\mathbf{t}$ at $\mathbf{n}$, that is, the tree obtained from $\mathbf{t}$ by deleting the subtrees rooted at the children of $\mathbf{n}$ is denoted by $\overline{\mathbf{t_n}}$;[3] and, for each $\sigma \in \Sigma$, the tree consisting of just one node that is labeled with σ is denoted by $\mathbf{t}(\sigma)$. The *depth* of a node $\mathbf{n}$ is the number of nodes on the path from $\mathbf{n}$ to the root ($\mathbf{n}$ included, root not included). The *height* of $\mathbf{n}$ is the number of nodes on the longest path from $\mathbf{n}$ to a leaf ($\mathbf{n}$ included, leaf not included). Hence, the depth of the root and the height of a leaf are zero. We denote the label of $\mathbf{n}$ in $\mathbf{t}$ by $\text{lab}_\mathbf{t}(\mathbf{n})$.

We end by introducing the following notation. When σ is a symbol in Σ and $\mathbf{t}_1, \ldots, \mathbf{t}_n$ are Σ-trees, then $\sigma(\mathbf{t}_1, \ldots, \mathbf{t}_n)$ is the Σ-tree graphically represented by

$$\begin{array}{ccc} & \sigma & \\ & \swarrow \cdots \searrow & \\ \mathbf{t}_1 & & \mathbf{t}_n. \end{array}$$

Note that in the above definitions there is no a priori bound on the number of children that a node may have. We refer to them as *unranked* trees.

2.3 Extended context-free grammars

Extended AGs are defined over extended context-free grammars which are defined as follows:

Definition 2.5 An *extended context-free grammar* (ECFG) is a tuple $G = (N, T, P, U)$, where

- T and N are disjoint finite non-empty sets, called the set of *terminals* and *non-terminals*, respectively;
- $U \in N$ is the *start symbol*; and
- P is a set of *productions* consisting of rules of the form $X \to r$ where $X \in N$ and r is a regular expression over $N \cup T$ such that $\varepsilon \notin L(r)$ and $L(r) \neq \emptyset$. Additionally, if $X \to r_1$ and $X \to r_2$ belong to P then $L(r_1) \cap L(r_2) \neq \emptyset$.

[3] Note that $\mathbf{t_n}$ and $\overline{\mathbf{t_n}}$ have $\mathbf{n}$ in common.

A *derivation tree* $\mathbf{t}$ over an ECFG G is a tree labeled with symbols from $N \cup T$ such that

- the root of $\mathbf{t}$ is labeled with U;
- for every interior node $\mathbf{n}$ with children $\mathbf{n}_1, \ldots, \mathbf{n}_m$ there exists a production $X \to r$ such that $\mathbf{n}$ is labeled with X, for $i = 1, \ldots, m$, $\mathbf{n}_i$ is labeled with X_i, and $X_1 \cdots X_m \in L(r)$; we say that $\mathbf{n}$ is *derived* by $X \to r$; and
- every leaf node is labeled with a terminal.

Note that derivation trees of ECFGs are *unranked*. Throughout this chapter we make the harmless technical assumption that the start symbol does not occur on the right-hand side of a production. The only place we make use of this convention is in the proof of Theorem 6.4.

2.4 Tree automata over unranked trees

We continue with the definition of nondeterministic bottom-up tree automata over unranked trees [11] by which the mechanism of extended AGs is inspired. Interestingly, these automata will also be used to obtain the exact complexity of testing non-emptiness and equivalence of extended AGs in Section 7.

Definition 2.6 A *nondeterministic bottom-up tree automaton* (NBTA) is a tuple $B = (Q, \Sigma, F, \delta)$, where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and δ is a function $Q \times \Sigma \to 2^{Q^*}$ such that $\delta(q, a)$ is a regular string language for every $a \in \Sigma$ and $q \in Q$. The semantics of B on a tree $\mathbf{t}$, denoted by $\delta^*(\mathbf{t})$, is defined inductively as follows: if $\mathbf{t}$ consists of only one node labeled with a then $\delta^*(\mathbf{t}) = \{q \mid \varepsilon \in \delta(q, a)\}$; if $\mathbf{t}$ is of the form

$$\begin{array}{ccc} & a & \\ \swarrow & \cdots & \searrow \\ \mathbf{t}_1 & & \mathbf{t}_n, \end{array}$$

then

$$\delta^*(\mathbf{t}) = \{q \mid \exists q_1 \in \delta^*(\mathbf{t}_1), \ldots, \exists q_n \in \delta^*(\mathbf{t}_n) \text{ and } q_1 \cdots q_n \in \delta(q, a)\}.$$

A tree $\mathbf{t}$ over Σ is *accepted* by the automaton B if $\delta^*(\mathbf{t}) \cap F \neq \emptyset$. The tree language *defined* by B, denoted by $L(B)$, consists of the trees accepted by

B. A tree language $\mathcal{T}$ is *recognizable* if there exists an NBTA B such that $\mathcal{T} = L(B)$.

Further, we say that B is *deterministic* when $\delta(q, a) \cap \delta(q', a) = \emptyset$ for every $a \in \Sigma$ and $q, q' \in Q$ with $q \neq q'$. We use the abbreviation DBTA to refer to such automata.

We represent the string languages $\delta(q, a)$ by NFAs. The *size* of B then is the sum of the sizes of Q, Σ, and the NFAs defining the transition function.

We will use the following notion in Section 7. A *state assignment* of B for a tree $\mathbf{t}$ is a mapping ρ from the nodes of $\mathbf{t}$ to Q. A state assignment is *valid* if for every node $\mathbf{n}$ of $\mathbf{t}$ of arity n, $\rho(\mathbf{n}1) \cdots \rho(\mathbf{n}n) \in \delta(\rho(\mathbf{n}), X)$, where $\mathbf{n}$ is labeled with X, and $\rho(\text{root}(\mathbf{t})) \in F$. Clearly, a tree $\mathbf{t}$ is accepted by $T_{\mathcal{F}}$ if and only if there exists a valid state assignment for t.

A detailed study of tree automata over unranked trees has been initiated by Brüggemann-Klein, Murata and Wood [11, 32]. Among many things, they show that DBTAs are as expressive as NBTAs and that the recognizable languages are closed under the Boolean operations.

Tree automata are defined over an arbitrary alphabet, but we consider derivation trees of ECFGs in this chapter. This seeming distinction can be dispensed with since we can always restrict an NBTA to the derivation trees of an ECFG as illustrated next. We point out that this lemma is well known for the ranked case with respect to CFGs [20].

Lemma 2.7 *Let $G = (N, T, P, U)$ be an ECFG and let B be an NTBA over $\Sigma \subseteq N \cup T$. Then there exists an NBTA B^G such that $L(B^G) = L(G) \cap L(B)$.*

Proof. We define an NBTA M such that $L(M) = L(G)$. Since recognizable tree languages are closed under the Boolean operations, we can then define B^G as an automaton accepting $L(M) \cap L(B)$.

Define $M = (Q, N \cup T, F, \delta)$, where $Q = T \cup P$, $F = \{U \to r \mid U \to r \in P\}$, and δ is defined as follows: for every $\sigma_1 \in T$ and $\sigma_2 \in T \cup N$,

$$\delta(\sigma_1, \sigma_2) := \begin{cases} \{\varepsilon\} & \text{if } \sigma_1 = \sigma_2; \\ \emptyset & \text{otherwise,} \end{cases}$$

and for every $X \to r \in P$ and $Y \in N \cup T$

$$\delta(X \to r, Y) := \begin{cases} L(r) & \text{if } X = Y; \\ \emptyset & \text{otherwise.} \end{cases}$$

■

The proof of the following lemma is a straightforward generalization of the ranked case (see, e.g., the survey paper by Vardi [46]).

Lemma 2.8 *Deciding whether the tree language accepted by an NBTA is non-empty is in PTIME.*

Proof. Let $B = (Q, \Sigma, F, \delta)$ be an NBTA. We inductively compute the set of *reachable* states R defined as follows: $q \in R$ iff there exists a tree $\mathbf{t}$ with $q \in \delta^*(\mathbf{t})$. Obviously, $L(B) \neq \emptyset$ if and only if $R \cap F \neq \emptyset$. Define for all $n > 0$,

$$\begin{aligned} R_1 &:= \{q \in Q \mid \exists a \in \Sigma : \varepsilon \in \delta(q, a)\}; \\ R_{n+1} &:= \{q \in Q \mid \exists a \in \Sigma : \delta(q, a) \cap R_n^* \neq \emptyset\}. \end{aligned}$$

Note that for all n, $R_n \subseteq R_{n+1} \subseteq Q$. Hence, $R_{|Q|} = R_{|Q|+1}$. Thus, define R as $R_{|Q|}$.

Clearly, R_1 can be computed in time linear in the size of B. Since testing non-emptiness of $\delta(q, a) \cap R_n^*$ can be done in time polynomial in the sum of the sizes of these (see, e.g., [25]), each R_{n+1} can be computed in time polynomial in the size of B. This concludes the proof of the lemma. ■

2.5 Two-way automata with a pebble

We conclude by introducing the following important device. A *two-way nondeterministic finite automaton with one pebble* is an NFA that can move in two directions over the input string and that has one pebble which it can lay down on the input string and pick back up later on. We refrain from giving a formal definition of such automata as we will only use them informally to describe our algorithmic computation. Blum and Hewitt [8] showed that such automata can only define regular languages. In the sequel, we will need the following stronger result obtained by Globerman and Harel [22, Proposition 3.2].

Proposition 2.9 *Every two-way nondeterministic finite automaton M with one pebble is equivalent to an NFA M' whose size is exponential in the size of M. In fact, the size of M' can be uniformly bounded by a function $|\Sigma| \cdot 2^{q(|S|)}$, where q is a polynomial, Σ is the alphabet, and S is the set of states of M. Additionally, M' can be constructed in time exponential in the size of M.*

3 Example

We give a small example introducing the important ideas for the definition of extended attribute grammars in the next section.

First, we briefly illustrate the mechanism of attribute grammars by giving an example of a Boolean-valued standard attribute grammar (BAG). The latter are studied by Neven and Van den Bussche [34, 37, 38]. As mentioned in the introduction, attribute grammars provide a mechanism for annotating the nodes of a tree with so-called "attributes", by means of so-called "semantic rules". A BAG assigns Boolean values by means of propositional logic formulas to attributes of nodes of input trees. Consider the CFG consisting of the productions $U \to AA$, $A \to a$, and $A \to b$. The following BAG selects the first A whenever the first A is expanded to an a and the second A is expanded to a b:

$$\begin{array}{ll} U \to AA & \mathit{select}(1) := \mathit{is\_a}(1) \wedge \neg \mathit{is\_a}(2); \\ A \to a & \mathit{is\_a}(0) := \mathit{true} \\ A \to b & \mathit{is\_a}(0) := \mathit{false} \end{array}$$

Here, the 1 in $\mathit{select}(1)$ indicates that the attribute *select* of the first A is being defined. Moreover, this attribute is true whenever the first A is expanded to an a (that is, $\mathit{is\_a}(1)$ should be true) and the second A is expanded to a b (that is, $\mathit{is\_a}(2)$ should be false). The other rules then define the attribute *is_a* in the obvious way. In the above, 0 refers to the left-hand side of the rule.

Consider the ECFG consisting of the sole rule $U \to (A+B)^*$. Suppose, we want to construct an attribute grammar selecting those A's that are preceded by an even number of A's and succeeded by an odd number of B's. Like above we will use rules defining the attribute *select*. This gives rise to two problems not present for BAGs : (i) U can have an unbounded number of children labeled with A which implies that an unbounded number of attributes should be defined; (ii) the definition of an attribute of an A depends on its siblings, whose number is again unbounded.

We resolve this in the following way. For (i), we just define *select* uniformly for each node that corresponds to the first position in the regular expression $(A + B)^*$. For (ii), we use regular languages as semantic rules rather than propositional formulas. The following extended AG expresses

the above query:

$$U \to (A+B)^* \qquad \begin{array}{l} select(1) := \langle \sigma_1 = \text{lab}, \sigma_2 = \text{lab}; \\ \quad R_{true} = (B^*AB^*AB^*)^*\#A^*BA^*(A^*BA^*BA^*)^*, \\ \quad R_{false} = (A+B+\#)^* - R_{true} \rangle. \end{array}$$

The 1 in $select(1)$ indicates that the attribute *select* is defined uniformly for every node corresponding to the *first* position in $(A+B)^*$. In the first part of the semantic rule, each σ_i lists the attributes of position i that will be used. Here, both for position 1 and 2 this is only the attribute lab which is a special attribute containing the label of the node. Consider the input tree $U(AAABBB)$. Then, to check, for instance, whether the third A is selected we enumerate the attributes mentioned in the first part of the rule and insert the symbol # before the node under consideration. This gives us the string

1	1	1	2	2	2	position in $(A+B)^*$
A	**A**	**#A**	**B**	**B**	**B**	
1	2	3	4	5	6	position in $AAABBB$

The attribute *select* of the third child will be assigned the value *true* since the above string belongs to R_{true}. Note that

$$(B^*AB^*AB^*)^* \text{ and } A^*BA^*(A^*BA^*BA^*)^*$$

define the set of strings with an even number of A's and with an odd number of B's, respectively. The above will be defined formally in the next section.

4 Attribute grammars over extended context-free grammars

We next define extended attribute grammars (extended AGs) over ECFGs whose attributes can take only values from a finite set D.

Proviso 4.1 Unless explicitly stated otherwise, we always assume an ECFG $G = (N, T, P, U)$. When we say tree we always mean derivation tree of G.

Definition 4.2 An *attribute grammar vocabulary* is a tuple $(D, A, \text{Syn}, \text{Inh})$, where

- D is a finite set of values called the *semantic domain*. We assume that D always contains the Boolean values 0 and 1;
- A is a finite set of symbols called *attributes*; we always assume that A contains the attribute *lab*;
- Syn and Inh are functions from $N \cup T$ to the powerset of $A - \{\text{lab}\}$ such that for every $X \in N$, $\text{Syn}(X) \cap \text{Inh}(X) = \emptyset$; for every $X \in T$, $\text{Syn}(X) = \emptyset$; and $\text{Inh}(U) = \emptyset$.

If $a \in \text{Syn}(X)$, we say that a is a *synthesized attribute of* X. If $a \in \text{Inh}(X)$, we say that a is an *inherited attribute of* X. We also agree that *lab* is an attribute of every X (this is a predefined attribute; for each node its value will be the label of that node). The above conditions express that an attribute cannot be a synthesized and an inherited attribute of the same grammar symbol, that terminal symbols do not have synthesized attributes, and that the start symbol does not have inherited attributes.

We now formally define the semantic rules of extended AGs. For a production $p = X \to r$, define $p(0) = X$, and for $i \in \{1, \ldots, |r|\}$, define $p(i) = r(i)$. We fix some attribute grammar vocabulary $(A, D, \text{Syn}, \text{Inh})$ in the following definitions.

Definition 4.3 1. Let $p = X \to r$ be a production of G and let a be an attribute of $p(i)$ for some $i \in \{0, \ldots, |r|\}$. The triple (p, a, i) is called a *context* if $a \in \text{Syn}(p(i))$ implies $i = 0$, and $a \in \text{Inh}(p(i))$ implies $i > 0$.

2. A *rule in the context* (p, a, i) is an expression of the form

$$a(i) := \langle \sigma_0, \ldots, \sigma_{|r|}; (R_d)_{d \in D} \rangle,$$

where

- for $j = \{0, \ldots, |r|\}$, σ_j is a sequence of attributes of $p(j)$;
- if $i = 0$, then, for each $d \in D$, R_d is a regular language over the alphabet D; and
- if $i > 0$, then, for each $d \in D$, R_d is a regular language over the alphabet $D \cup \{\#\}$.

For all $d, d' \in D$, if $d \neq d'$ then $R_d \cap R_{d'} = \emptyset$. Further, if $i = 0$ then $\bigcup_{d \in D} R_d = D^*$. If $i > 0$ then $\bigcup_{d \in D} R_d$ should contain all strings over D with exactly one occurrence of the symbol #. Note that a R_d is allowed to contain strings with several occurrences of #. We always assume that $\# \notin D$.

An extended AG is then defined as follows:

Definition 4.4 An *extended attribute grammar (extended AG)* $\mathcal{F}$ consists of an attribute grammar vocabulary, together with a mapping assigning to each context a rule in that context.

It will always be understood which rule is associated to which context. We illustrate the above definitions with an example.

Example 4.5 In Figure 3 an example of an extended AG $\mathcal{F}$ is depicted over the ECFG of Figure 2. Recall that every grammar symbol has the attribute lab; for each node this attribute has the label of that node as value. We have Syn(Word) = $\{king, lord\}$, Syn(Verse) = $\{king\_lord\}$, Syn(Poem) = $\{result\}$, and Inh(Poem) = $\{first\}$. The grammar symbols DB, a, ... , z, Verse, and Word have no attributes apart from lab. The semantics of this extended AG will be explained below. Here,

$$D = \{0, 1, \mathrm{a}, \ldots, \mathrm{z}, \mathrm{DB}, \mathrm{Poem}, \mathrm{Verse}, \mathrm{Word}\}.$$

We use regular expressions to define the languages R_1; for the first rule, R_0 is defined as $(D \cup \{\#\})^* - R_1$; for all other rules, R_0 is defined as $D^* - R_1$; those R_d that are not specified are empty; ε stands for the empty sequence of attributes. ■

$\mathrm{DB} \to \mathrm{Poem}^+$	$first(1) := \langle \sigma_0 = lab, \sigma_1 = lab; R_1 = \mathrm{DB}\#\mathrm{Poem}^+ \rangle$
$\mathrm{Poem} \to \mathrm{Verse}^+$	$result(0) := \langle \sigma_0 = first, \sigma_1 = king\_lord;$ $R_1 = 1(1+0)^* + 0(1(1+0))^*(1+\varepsilon) \rangle$
$\mathrm{Verse} \to \mathrm{Word}^+$	$king\_lord(0) := \langle \sigma_0 = \varepsilon, \sigma_1 = (king, lord);$ $R_1 = (0+1)^* + 1 + (0+1)^* \rangle$
$\mathrm{Word} \to (a + \ldots + z)^+$	$king(0) := \langle \sigma_0 = \varepsilon, \sigma_1 = lab, \ldots, \sigma_{26} = lab; R_1 = \{\mathrm{king}\} \rangle$
	$lord(0) := \langle \sigma_0 = \varepsilon, \sigma_1 = lab, \ldots, \sigma_{26} = lab; R_1 = \{\mathrm{lord}\} \rangle$

Figure 3: Example of an extended AG.

The semantics of an extended AG is that it defines attributes of the nodes of derivation trees of the underlying grammar G. This is formalized next.

Definition 4.6 If $\mathbf{t}$ is a derivation tree of G then a *valuation* v *of* $\mathbf{t}$ is a function that maps each pair $(\mathbf{n}, a)$, where $\mathbf{n}$ is a node in $\mathbf{t}$ and a is an attribute of the label of $\mathbf{n}$, to an element of D, and that maps for every $\mathbf{n}$, $v((lab, \mathbf{n}))$ to the label of $\mathbf{n}$.

In the sequel, for a pair $(\mathbf{n}, a)$ as above we will use the more intuitive notation $a(\mathbf{n})$. To define the semantics of $\mathcal{F}$ we first need the following definition. If $\sigma = a_1 \cdots a_k$ is a sequence of attributes and $\mathbf{n}$ is a node of $\mathbf{t}$, then define $\sigma(\mathbf{n})$ as the sequence of attribute-node pairs $\sigma(\mathbf{n}) = a_1(\mathbf{n}) \cdots a_k(\mathbf{n})$.

Definition 4.7 Let $\mathbf{t}$ be a derivation tree, $\mathbf{n}$ a node of $\mathbf{t}$, and a an attribute of the label of $\mathbf{n}$.

Synthesized Let $\mathbf{n}$ be a node of arity n derived by $p = X \to r$, and let $\langle \sigma_0, \ldots, \sigma_{|r|}; (R_d)_{d \in D} \rangle$ be the rule associated to the context $(p, a, 0)$. Define for $l \in \{1, \ldots, m\}$, $j_l = \mathrm{pos}_r(l, w)$, where w is the string formed by the labels of the children of $\mathbf{n}$. Then define $W(a(\mathbf{n}))$ as the sequence

$$\sigma_0(\mathbf{n}) \cdot \sigma_{j_1}(\mathbf{n}1) \cdots \sigma_{j_m}(\mathbf{n}m).$$

For each d, we denote the language R_d associated to $a(\mathbf{n})$ by $R_d^{a(\mathbf{n})}$.

Inherited Let $\mathbf{n}_1, \ldots, \mathbf{n}_{k-1}$ be the left siblings, $\mathbf{n}_{k+1}, \ldots, \mathbf{n}_m$ be the right siblings, and $\mathbf{n}_0$ be the parent of $\mathbf{n}$. Let $\mathbf{n}_0$ be derived by $p = X \to r$, and define for $l \in \{1, \ldots, m\}$, $j_l = \mathrm{pos}_r(l, w)$, where w is the string formed by the labels of the children of $\mathbf{n}_0$. Let $\langle \sigma_0, \ldots, \sigma_{|r|}; (R_d)_{d \in D} \rangle$ be the rule associated to the context (p, a, j_k). Now define $W(a(\mathbf{n}))$ as the sequence

$$\sigma_0(\mathbf{n}_0) \cdot \sigma_{j_1}(\mathbf{n}_1) \cdots \sigma_{j_{k-1}}(\mathbf{n}_{k-1}) \cdot \# \cdot \sigma_{j_k}(\mathbf{n}) \cdots \sigma_{j_{k+1}}(\mathbf{n}_{k+1}) \sigma_{j_m}(\mathbf{n}_m).$$

For each d, we denote the language R_d associated to $a(\mathbf{n})$ by $R_d^{a(\mathbf{n})}$.

If v is a valuation then define $v(W(a(\mathbf{n})))$ as the string obtained from $W(a(\mathbf{n}))$ by replacing each $b(\mathbf{m})$ in $W(a(\mathbf{n}))$ by $v(b(\mathbf{m}))$. Note that the empty sequence is just replaced by the empty string.

We are now ready to define the semantics of an extended AG $\mathcal{F}$ on a derivation tree.

Definition 4.8 Given an extended AG $\mathcal{F}$ and a derivation tree $\mathbf{t}$, we define a sequence of partial valuations $(\mathcal{F}_j)_{j\geq 0}$ as follows:

1. $\mathcal{F}_0(\mathbf{t})$ is the valuation that maps, for every node $\mathbf{n}$, $\mathrm{lab}(\mathbf{n})$ to the label of $\mathbf{n}$ and is undefined everywhere else;

2. for $j > 0$, if $\mathcal{F}_{j-1}(\mathbf{t})$ is defined on all $b(\mathbf{m})$ occurring in $W(a(\mathbf{n}))$ then

$$\mathcal{F}_j(\mathbf{t})(a(\mathbf{n})) = d,$$

where $\mathcal{F}_{j-1}(W(a(\mathbf{n}))) \in R_d^{a(\mathbf{n})}$. Note that this is well defined.

If for every $\mathbf{t}$ there is an l such that $\mathcal{F}_l(\mathbf{t})$ is totally defined (this implies that $\mathcal{F}_l(\mathbf{t}) = \mathcal{F}_{l-1}(\mathbf{t})$) then we say that $\mathcal{F}$ is *non-circular*. Obviously, non-circularity is an important property. In the next section we show that it is decidable whether an extended AG is non-circular. Therefore, we can state the following proviso.

Proviso 4.9 In the sequel we always assume an extended AG to be non-circular.

Definition 4.10 The valuation $\mathcal{F}(\mathbf{t})$ equals $\mathcal{F}_l(\mathbf{t})$ with l such that $\mathcal{F}_l(\mathbf{t}) = \mathcal{F}_{l+1}(\mathbf{t})$.

Proviso 4.11 Whenever we say query, we always mean *unary* query.

An extended AG $\mathcal{F}$ can be used in a simple way to express queries. Among the attributes in the vocabulary of $\mathcal{F}$, we designate some attribute *result*, and define:

Definition 4.12 An extended AG $\mathcal{F}$ expresses the query $\mathcal{Q}$ defined by

$$\mathcal{Q}(\mathbf{t}) = \{\mathbf{n} \mid \mathcal{F}(\mathbf{t})(\mathit{result}(\mathbf{n})) = 1\},$$

for every tree $\mathbf{t}$.

Example 4.13 Recall the extended AG $\mathcal{F}$ of Figure 3. This extended AG selects the first poem and every poem that has the strings king or lord in every other verse starting from the first one. In Figure 4 an illustration is given of the result of $\mathcal{F}$ on a derivation tree $\mathbf{t}$. At each node $\mathbf{n}$, we show the values $\mathcal{F}(W(a(\mathbf{n})))$ and $\mathcal{F}(\mathbf{t})(a(\mathbf{n}))$. We abbreviate $a(\mathbf{n})$ by a, *king* by *k*, *lord* by *l*, and *king_lord* by *k_l*.

The definition of the inherited attribute *first* indicates how the use of # can distinguish in a uniform way between different occurrences of the grammar symbol Poem. This is only a simple example. In the next section we show that extended AGs can express all queries definable in MSO. Hence, they can also specify all relationships between siblings definable in MSO.

The language R_1 associated to *result* (cf. Figure 3), contains those strings representing that the current Poem is the first one, or representing that for every other verse starting at the first one the value of the attribute *king_lord* is 1. ■

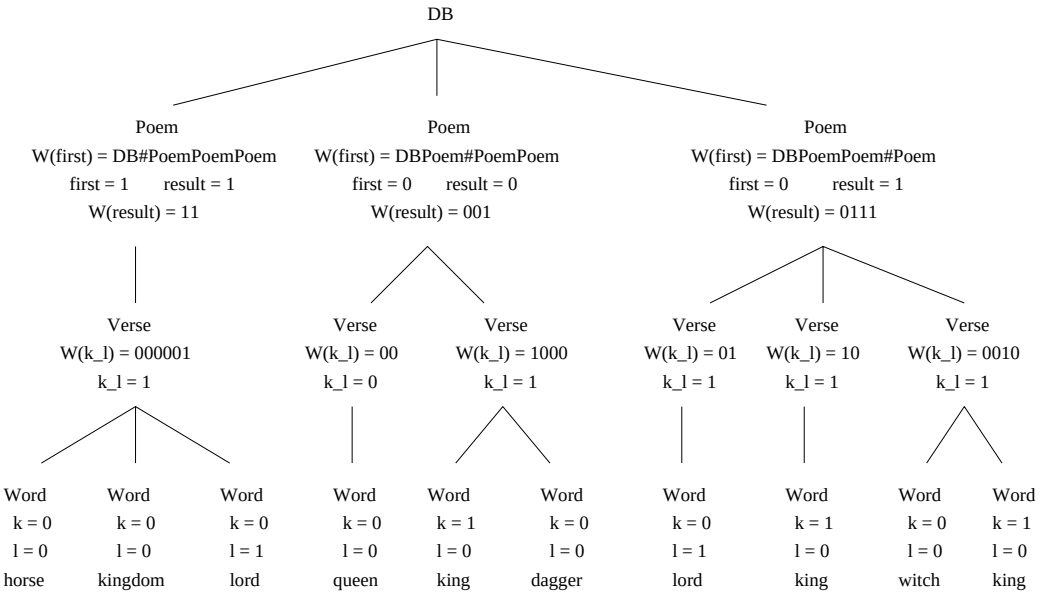

Figure 4: A derivation tree and its valuation as defined by the extended AG in Figure 3.

The *size* of an extended AG is the sum of the sizes of the attribute grammar vocabulary, the ECFG and the size of the semantic rules where we

represent the regular languages R_d by NFAs.

5 Non-circularity

In this section we show that it is decidable whether an extended AG is non-circular. In particular, we show that deciding non-circularity is in EXPTIME. As it is well known that deciding non-circularity of *standard* AGs is complete for EXPTIME [26], this result indicates that going from ranked to unranked trees does not increase the complexity of the non-circularity problem.

We first make the following remark indicating that testing non-circularity for extended AGs is slightly more subtle than for standard AGs.

Remark 5.1 Not all the specified attributes in a semantic rule are always used. Indeed, consider the grammar with productions $C \to A + B$, $A \to c$ and $B \to c$. Let $\mathcal{F}$ be an extended AG where the inherited attribute a of A and B is defined in the context $(C \to A + B, a, 1)$ as

$$a(1) := \langle \sigma_0 = \varepsilon, \sigma_1 = \varepsilon, \sigma_2 = a; R_1 \rangle,$$

and in the context $(C \to A + B, a, 2)$ as

$$a(2) := \langle \sigma_0 = \varepsilon, \sigma_1 = a, \sigma_2 = \varepsilon; R_1 \rangle.$$

At first sight $\mathcal{F}$ seems circular. This is, however, not the case since A and B never occur simultaneously in a derivation tree. Consider for example the tree graphically represented as

$$\begin{array}{c} C \\ \downarrow \\ A \\ \downarrow \\ c. \end{array}$$

If the label of $\mathbf{n}$ is A then $W(a(\mathbf{n}))$ is the empty sequence and consequently $\mathcal{F}(a(\mathbf{n})) = 1$ if and only if the empty string belongs to R_1. ■

A naive approach to testing non-circularity is to transform an extended AG $\mathcal{F}$ into a standard AG $\mathcal{F}'$ such that $\mathcal{F}$ is non-circular if and only if $\mathcal{F}'$ is non-circular and then use the known exponential algorithm on $\mathcal{F}'$. We can namely always find an integer N (polynomially depending on $\mathcal{F}$) such that we

only have to test non-circularity of $\mathcal{F}$ on trees of rank N. Unfortunately, this approach can increases the size of the AG more than polynomially. Indeed, a production $X \to (a+b)\cdots(a+b)$ (n times), for instance, has to be translated to the set of productions $\{X \to w \mid w \in \{a,b\}^* \wedge |w| = n\}$. So, the complexity of the naive algorithm is double exponential time. Therefore, we abandon this approach and give a different algorithm whose complexity is in EXPTIME.

To this end, we first generalize the tree walking automata of Bloem and Engelfriet [7] to unranked trees. In particular, we show that for each extended AG $\mathcal{F}$, there exists a tree walking automata $W_{\mathcal{F}}$ such that $\mathcal{F}$ is non-circular if and only if $W_{\mathcal{F}}$ does not cycle on any input tree. Moreover, $W_{\mathcal{F}}$ can be constructed in time polynomial in the size of $\mathcal{F}$. We thus obtain our result by showing that testing whether a tree walking automaton cycles is in EXPTIME.

Definition 5.2 A *nondeterministic tree walking automaton* is a tuple $W = (Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states,
- Σ is an alphabet,
- $q_0 \in Q$ is the start state,
- $F \subseteq Q$ is the set of final states, and
- $\delta \subseteq Q \times \Sigma \times Q \times \{\downarrow_{\text{first}}, \downarrow_{\text{last}}, \rightarrow, \leftarrow, \uparrow, \text{stay}\}$ is the transition relation.

Intuitively, a tree walking automaton walks over the tree starting at the root. To make sure that the automaton cannot fall off the tree, we augment input trees with the boundary symbols $\leftarrow$, $\rightarrow$, $\downarrow$, and $\uparrow$. For example, the tree $\mathbf{t} := a(b,c)$ augmented with boundary symbols is defined as

$$\text{bound}(\mathbf{t}) \ := \downarrow(\rightarrow, a(\rightarrow, b(\uparrow), c(\uparrow), \leftarrow), \leftarrow),$$

or more graphically:

$$\begin{array}{ccccc} & & \downarrow & & \\ & & \rightarrow a \leftarrow & & \\ \rightarrow b & & & & c \leftarrow \\ \uparrow & & & & \uparrow \end{array} .$$

We use another auxiliary notion. We define $b(\mathbf{t})$ as bound$(\mathbf{t})$ without boundary symbols for root$(\mathbf{t})$. That is, $b(\mathbf{t})$ is the tree graphically represented by

$$\begin{array}{c} a \\ \rightarrow b \;\; c \leftarrow \\ \uparrow \;\; \uparrow \end{array} .$$

A perhaps more elegant solution is to have a separate transition function for the root node, internal nodes and leaf nodes. But since this last approach terribly complicates the proof of the next lemma we just stick to the tree representation with boundary symbols.

We still have to explain the semantics of tree walking automaton. Depending on the current state and on the label at the current node, the transition relation determines in which direction the automaton can move and into which states it can change. The possible directions w.r.t. the current node are: go to the first child, the last child, the left sibling, the right sibling, or the parent, or stay at the current node. Of course, we have the obvious restrictions that W can only move to the left, right, down and up, when it reads the symbols $\leftarrow$, $\rightarrow$, $\downarrow$, and $\uparrow$, respectively.

The automaton accepts an input tree when there exists a walk started at the root in the start state that again reaches the root node in a final state. We make this more precise. A *configuration* of W on a tree $\mathbf{t}$ is a pair $(\mathbf{n}, q)$ where $\mathbf{n}$ is a node of bound$(\mathbf{t})$ and $q \in Q$. The *start* configuration is $(\text{root}(\mathbf{t}), q_0)$, and each $(\text{root}(\mathbf{t}), q)$ with $q \in F$ is an *accepting* configuration. A *walk* of W on $\mathbf{t}$ is a (possibly infinite) sequence of configurations $c_1 c_2 c_3 \cdots$ where c_1 is the start configuration and each c_{i+1} can be reached from c_i by making one transition. The latter is defined in the obvious way. A walk is *accepting* when it is finite and the last configuration is an accepting one. Finally, W *accepts* $\mathbf{t}$ when there exists an accepting walk of W on $\mathbf{t}$. However, we will not need this latter definition any further, as we are only interested in the existence of infinite walks.

We need the following definition to state the next lemma.

Definition 5.3 A nondeterministic tree walking automaton *cycles* if there is a tree on which it has an infinite walk.

Lemma 5.4 *Deciding whether a nondeterministic walking tree automaton cycles, is in EXPTIME.*

Proof. Let $W = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic tree walking automaton. For a tree $\mathbf{t}$ define the *behavior relation* of W on $\mathbf{t}$ as the relation $f_{\mathbf{t}}^W \subseteq Q \times (Q \cup \{\#\})$ as follows. For each $q, q' \in Q$,

1. $f_{\mathbf{t}}^W(q, q')$ if there exists a walk of W starting at the root of $\mathbf{t}$ in state q that again returns at the root in state q' with the additional requirement that W is not allowed to move to the left sibling, the right sibling or the parent of the root (recall these are labeled with $\rightarrow$, $\leftarrow$, and $\downarrow$, respectively) during this walk; in brief, W is only allowed to walk in $b(\mathbf{t})$;

2. $f_{\mathbf{t}}^W(q, \#)$ if there is an infinite walk of W starting at the root in state q, again with the additional requirement that W is not allowed to move to the left sibling, the right sibling or the parent of the root during this walk.

The additional requirement mentioned in both of the above cases is needed because we want to compute behavior relations of nodes in a tree, in terms of the behavior relations at the children of those nodes. Therefore, the behavior relations of the subtrees should only be defined by computations that do not leave those subtrees.

Let $f \subseteq Q \times (Q \cup \{\#\})$ be a relation and let $\sigma \in \Sigma$. Then, we say that (f, σ) is *satisfiable* whenever there exists a tree $\mathbf{t}$ with $f_{\mathbf{t}}^W = f$ and the label of root($\mathbf{t}$) is σ. We refer to the tuples (f, σ) as *behavior tuples*. It now suffices to compute the set of all satisfiable behavior tuples to decide whether W is cycling. To see this, we first introduce the following relations that determine the behavior of W when it encounters the boundary of a tree at its root. Define the relations $\delta^\sigma_{\substack{\rightarrow \\ \leftarrow}}$, $\delta^\sigma_{\substack{\leftarrow \\ \rightarrow}}$, and $\delta^\sigma_{\uparrow\downarrow}$, as follows: for each $q, q' \in Q$,

- $\delta^\sigma_{\substack{\rightarrow \\ \leftarrow}}(q, q')$ iff there exists a q'' such that $\delta(q, \sigma, q'', \rightarrow)$ and $\delta(q'', \leftarrow, q', \leftarrow)$;
- $\delta^\sigma_{\substack{\leftarrow \\ \rightarrow}}(q, q')$ iff there exists a q'' such that $\delta(q, \sigma, q'', \leftarrow)$ and $\delta(q'', \rightarrow, q', \rightarrow)$;
- $\delta^\sigma_{\uparrow\downarrow}(q, q')$ iff there exists a q'' such that $\delta(q, \sigma, q'', \uparrow)$ and $\delta(q'', \downarrow, q', \downarrow_{\text{first}})$;

We define States$(f_1, \ldots, f_n, q) \subseteq Q \cup \{\#\}$ as the set of states reachable from q by applying relations in $f_1, \ldots, f_n$. Further, if the relations introduce cycling then $\#$ also belongs to States$(f_1, \ldots, f_n, q)$. Formally:

1. States$(f_1, \ldots, f_n, q)$ is the smallest set of states containing q such that if $q' \in$ States$(f_1, \ldots, f_n, q)$ and $f_i(q', q'')$ then $q'' \in$ States$(f_1, \ldots, f_n, q)$ (note that $q'' \in Q \cup \{\#\}$);

2. additionally, if $q_1, \ldots, q_{m+1} \in \text{States}(f_1, \ldots, f_n, q)$, $q_1 = q_{m+1}$, and for $i = 1, \ldots, m$, there is a j_i with $f_{j_i}(q_i, q_{i+1})$, then $\# \in \text{States}(f_1, \ldots, f_n, q)$.

So, W is cycling whenever there exists a satisfiable behavior tuple (f, σ) such that $\# \in \text{States}(f, \delta^\sigma_{\overrightarrow{\leftarrow}}, \delta^\sigma_{\overleftarrow{\rightarrow}}, \delta^\sigma_{\uparrow\downarrow}, q_0)$. This just says that an infinite walk can be reached from the start state q_0.

To reduce the complexity of our algorithm we make use of a weaker notion of satisfiability. We say that a behavior tuple (f, σ) is *weakly satisfiable* whenever there exists a satisfiable behavior tuple (g, σ) such that $f \subseteq g$. Note that every satisfiable tuple is also weakly satisfiable. If g is witnessed by $\mathbf{t}$ then we say that f is *weakly witnessed* by $\mathbf{t}$. Further, let S be the set of all weakly satisfiable behavior tuples. Then W is cycling whenever there exists a behavior tuple $(f, \sigma) \in S$ such that $\# \in \text{States}(f, \delta^\sigma_{\overrightarrow{\leftarrow}}, \delta^\sigma_{\overleftarrow{\rightarrow}}, \delta^\sigma_{\uparrow\downarrow}, q_0)$.

In Figure 5, we give an algorithm computing S. In this algorithm, C is initialized by the set of satisfiable behavior tuples witnessed by 1-node trees. Hereafter, the algorithm tests for each behavior tuple (f, σ) whether it can be obtained by combining behavior tuples in C and adds (f, σ) to C if this is the case. To this end, we use an automaton $M_{f,\sigma}$ over the alphabet consisting of all behavior tuples. In particular, if (f_1, σ_1), $\ldots$, (f_n, σ_n) are weakly satisfiable and $(f_1, \sigma_1) \cdots (f_n, \sigma_n) \in L(M_{f,\sigma})$ then (f, σ) is weakly satisfiable. Moreover, if each (f_i, σ_i) is weakly witnessed by $\mathbf{t}_i$, then (f, σ) is weakly witnessed by $\sigma(\mathbf{t}_1, \ldots, \mathbf{t}_n)$. From this it follows that all tuples in C are weakly satisfiable. The converse can be shown by induction on the minimal height of the trees weakly witnessing the weakly satisfiable behavior tuples. It follows that after completion of the algorithm $C = S$. Since the size of each $M_{f,\sigma}$ will be exponential in the size of W, the test $L(M_{f,\sigma}) \cap C^* \neq \emptyset$ can be done in exponential time. As there are only exponentially many behavior tuples, the REPEAT loop will iterate at most an exponential number of times. Thus, the total execution time of the algorithm will be exponential in the size of W.

It remains to explain the construction of $M_{f,\sigma}$. First, we define a nondeterministic two-way string automaton $M'_{f,\sigma}$ with one pebble whose size is polynomial in the size of W. By Proposition 2.9, $M'_{f,\sigma}$ is equivalent to a one-way nondeterministic automaton whose size is only exponential in the size of $M'_{f,\sigma}$. We then define $M_{f,\sigma}$ as the latter automaton and the proof is finished.

On input $(f_1, \sigma_1) \cdots (f_n, \sigma_n)$, $M'_{f,\sigma}$ works as follows. We only have to

Input: W
% Initialization
for each behavior tuple (f, σ) **do**
    construct $M_{f,\sigma}$
$C := \{(f, \sigma) \mid f \subseteq f^W_{\mathbf{t}(\sigma)}, \sigma \in \Sigma\}$
% Main loop
repeat
    for each $(f, \sigma) \notin C$ **do**
        if $L(M_{f,\sigma}) \cap C^* \neq \emptyset$
        then $C := C \cup \{(f, \sigma)\}$
until no more changes occur

Figure 5: An algorithm computing the set S of weakly satisfiable behavior tuples.

consider the case where each (f_i, σ_i) is weakly satisfiable. Therefore, let each (f_i, σ_i) be weakly witnessed by $\mathbf{t}_i$.

1. For each $q, q' \in Q$ for which $f(q, q')$, the automaton $M'_{f,\sigma}$ has to check whether there exists a walk starting at the root of $\sigma(\mathbf{t}_1, \ldots, \mathbf{t}_n)$ in state q that again reaches the root in state q'. However, $M'_{f,\sigma}$ does not need to know the tree $\sigma(\mathbf{t}_1, \ldots, \mathbf{t}_n)$: $M'_{f,\sigma}$ just guesses this path using the f_i's. That is, $M'_{f,\sigma}$ starts in state q at the root. If W, for example, decides to move to the last child in state q_1, then $M'_{f,\sigma}$ walks to the last position of the string $(f_1, \sigma_1) \cdots (f_n, \sigma_n)$ arriving there in state q_1. Further, if $M'_{f,\sigma}$ arrives at a position labeled with (f_i, σ_i) and W decides to enter the subtree below this position, then $M'_{f,\sigma}$ just examines the relation f_i to see in which states it can return. If W makes a move to, say, the right sibling in state q_2, then $M'_{f,\sigma}$ just makes a right move to state q_2. If $M'_{f,\sigma}$ succeeds in reaching the root in state q', then it considers the next pair of states q_1 and q'_1 for which $f(q_1, q'_1)$. If all pairs are checked, $M'_{f,\sigma}$ moves to the next step. Clearly, $M'_{f,\sigma}$ only needs a number of states that is polynomial in the size of W.

2. For every $q \in Q$ such that $f(q, \#)$, $M'_{f,\sigma}$ has to verify the existence

of an infinite walk on $\sigma(\mathbf{t}_1, \ldots, \mathbf{t}_n)$ starting from state q at the root. This can happen in two ways. The first possibility is that W gets into a cycle in one of the subtrees $\mathbf{t}_1, \ldots, \mathbf{t}_n$, say $\mathbf{t}_i$. This can be detected, like in the previous case, by simply guessing a walk reaching position i of the input string $(f_1, \sigma_1) \cdots (f_n, \sigma_n)$ in a state q' such that $f_i(q', \#)$. The second possibility is that W can walk forever on the children of the root. We use the pebble to detect this: $M'_{f,\sigma}$ now just guesses a walk of W using the relations $f_1, \ldots, f_n$ as explained above and nondeterministically puts down its pebble on a position of $(f_1, \sigma_1) \cdots (f_n, \sigma_n)$, memorizes the current state, and proceeds its walk. When the automaton reaches the pebble again in the memorized state, which means that W indeed has reached a cycle, $M'_{f,\sigma}$ checks the next state $q' \in Q$ for which $f(q', \#)$. If all pairs are checked, $M'_{f,\sigma}$ accepts. Clearly, $M'_{f,\sigma}$ only needs a number of states that is polynomial in the size of W. ■

We note that the idea of computing behavior functions (as opposed to relations) in a bottom-up way was already used by Schwentick and the present author [36] to obtain upper bounds on the complexity of various optimization problems for query automata.

Next, we define a tree walking automaton $W_{\mathcal{F}}$ for an extended AG $\mathcal{F}$ such that $W_{\mathcal{F}}$ cycles if and only if $\mathcal{F}$ is circular. The idea is that on input $\mathbf{t}$, $W_{\mathcal{F}}$ follows all possible paths in the dependency graph[4] of $\mathcal{F}$ for $\mathbf{t}$. Hence, $W_{\mathcal{F}}$ will terminate on $\mathbf{t}$ if and only if this dependency graph is acyclic. This idea is similar in spirit to a result by Maneth and the present author [31] where a standard attribute grammar is transformed to a $\mathcal{DTL}$ program such that the latter terminates on every input if and only if the former is non-circular. Here, the complication arises from the fact that we have to deal with extended AGs over *unranked* trees rather than with standard AGs over *ranked* trees.

Theorem 5.5 *Deciding non-circularity of extended AGs is in EXPTIME.*

[4]The *dependency graph* $\mathcal{D}_{\mathcal{F}}(\mathbf{t})$ of $\mathcal{F}$ for a derivation tree $\mathbf{t}$ is defined as follows. Its nodes are all $a(\mathbf{n})$, such that $\mathbf{n}$ is a node of $\mathbf{t}$ and a is an attribute of the label of $\mathbf{n}$. Further, there is an edge from $a(\mathbf{n})$ to $b(\mathbf{m})$ if and only if $a(\mathbf{n}) \in W(b(\mathbf{m}))$ (cf. Definition 4.7). Clearly, $\mathcal{F}$ is well-defined on $\mathbf{t}$ if and only if $\mathcal{D}(\mathbf{t})$ contains no cycle. Hence, $\mathcal{F}$ is non-circular if and only if there does not exist a $\mathbf{t}$ such that $\mathcal{D}_{\mathcal{F}}(\mathbf{t})$ is cyclic.

Proof. Let $\mathcal{F}$ be an extended AG with attribute set A and semantic domain D. For ease of exposition we assume that all grammar symbols have all attributes, i.e., for every $X \in N \cup T$, $\mathrm{Inh}(X) \cup \mathrm{Syn}(X) = A$.

We construct a tree walking automaton $W_{\mathcal{F}}$ such that $W_{\mathcal{F}}$ cycles if and only if $\mathcal{F}$ is circular. Rather than letting $W_{\mathcal{F}}$ work over derivation trees of G, we let it work on the set of all trees over the alphabet $(N \cup T) \times (P \cup T) \times (P \cup \{U\}) \times \{1, \ldots, m\}$ where $m = \max\{|r| \mid X \to r \in P\}$. That is, m denotes the maximal number of positions of a regular expression in a production of P.

The automaton $W_{\mathcal{F}}$ first checks the consistency of the labelings. That is, for each node $\mathbf{n}$ of the input tree labeled with (σ, p_1, p_2, i),

1. if $p_1 \in T$ then $\sigma = p_1$ and $\mathbf{n}$ should be a leaf; if $p_1 = X \to r \in P$ then $\sigma = X$ and $\mathbf{n}$ should be derived by p_1;

2. if $p_2 = U$ then $\mathbf{n}$ should be the root; if $p_2 \in P$ then the parent of $\mathbf{n}$ should be derived with p_2; and

3. if the parent $\mathbf{p}$ of $\mathbf{n}$ is derived by $X \to r$, w is the string formed by the children of $\mathbf{p}$, and $\mathbf{n}$ is the j-th child of $\mathbf{p}$, then $\mathrm{pos}_r(j, w) = i$.

The automaton checks this in the following way. It makes a depth first traversal of the tree. At each node $\mathbf{n}$ labeled with (σ, p_1, p_2, i) it can check (1) by first checking whether the current node is a leaf, and if not, by simulating the NFA for r on the children of $\mathbf{n}$ where $p_1 = X \to r$. Only when the NFA accepts it moves to the next node in the depth first traversal. To check (2), $W_{\mathcal{F}}$ makes another depth first traversal of the tree. It first checks whether the root is labeled with $(U, p, U, 1)$. Next, for each internal node $\mathbf{n}$ labeled with (σ, p_1, p_2, i) it checks whether every child of $\mathbf{n}$ has p_1 in the third component of its label. Finally, (3) is checked by making a third depth first traversal through the tree. Arriving at a node $\mathbf{n}$ derived by $X \to r$, the automaton simulates the NFA M_r of Lemma 2.3 to check the fourth component of every label.

If all this succeeds then $W_{\mathcal{F}}$ nondeterministically walks to a node and chooses an attribute a which it keeps in its state. Now, suppose $W_{\mathcal{F}}$ arrives at a node $\mathbf{n}$ labeled with (X, p_1, p_2, j) with the attribute a in its state. We distinguish two cases.

1. a is a synthesized attribute of X: Let $a(0) := \langle \sigma_0, \ldots, \sigma_{|r|}; (R_d)_{d \in D} \rangle$ be the rule in the context $(p_1, a, 0)$. Then $W_{\mathcal{F}}$ nondeterministically

chooses an attribute b in a σ_i and replaces a in its state with b. If $i = 0$ then $W_{\mathcal{F}}$ just stays at the current node. If $i > 0$ then $W_{\mathcal{F}}$ walks nondeterministically to a child of the current node having i as the last component of its label.

2. a is an inherited attribute of X: Let $a(j) := \langle \sigma_0, \ldots, \sigma_{|r|}; (R_d)_{d \in D} \rangle$ be the rule in the context (p_2, a, j). Then $W_{\mathcal{F}}$ nondeterministically chooses an attribute b in a σ_i and replaces a in its state with b. If $i = 0$ then $W_{\mathcal{F}}$ walks to the parent of $\mathbf{n}$. If $i > 0$ then $W_{\mathcal{F}}$ walks nondeterministically to a sibling of $\mathbf{n}$ having i as the last component of its label (or possibly stays at $\mathbf{n}$ if $i = j$).

Clearly, $W_{\mathcal{F}}$ is cycling if and only if $\mathcal{F}$ is circular. Moreover, $W_{\mathcal{F}}$ can be constructed in time polynomial in the size of $\mathcal{F}$. ■

Since deciding non-circularity for standard attribute grammars is also hard for EXPTIME, we obtain that testing whether a nondeterministic tree walking automaton cycles is EXPTIME-complete.

6 Expressiveness of extended AGs

We characterize the expressiveness of extended AGs as the queries definable in monadic second-order logic. *Monadic second-order logic* (MSO) allows the use of *set variables* ranging over sets of nodes of a tree, in addition to the individual variables ranging over the nodes themselves as provided by first-order logic. We will assume some familiarity with this logic and refer the unfamiliar reader to the book of Ebbinghaus and Flum [19] or the chapter by Thomas [45].

A derivation tree $\mathbf{t}$ can be viewed naturally as a finite relational structure (in the sense of mathematical logic [19]) over the binary relation symbols $\{E, <\}$ and the unary relation symbols $\{O_a \mid a \in N \cup T\}$. The domain of $\mathbf{t}$, viewed as a structure, equals the set of nodes of $\mathbf{t}$. The relation E in $\mathbf{t}$ equals the set of pairs $(\mathbf{n}, \mathbf{n}')$ such that $\mathbf{n}'$ is a child of $\mathbf{n}$ in $\mathbf{t}$. The relation $<$ in $\mathbf{t}$ equals the set of pairs $(\mathbf{n}, \mathbf{n}')$ such that $\mathbf{n}' \neq \mathbf{n}$, $\mathbf{n}'$ and $\mathbf{n}$ are children of the same parent and $\mathbf{n}'$ is a child occurring after $\mathbf{n}$. The set O_a in $\mathbf{t}$ equals the set of a-labeled nodes of $\mathbf{t}$.

MSO can be used in the standard way to define queries. If $\varphi(x)$ is an MSO-formula, then φ defines the query $\mathcal{Q}$ defined by $\mathcal{Q}(\mathbf{t}) := \{\mathbf{n} \mid \mathbf{t} \models \varphi[\mathbf{n}]\}$.

We start with the easy direction.

Lemma 6.1 *Every query expressible by an extended AG is definable in MSO.*

Proof. Let $\mathcal{F}$ be an extended AG. We say that an arbitrary total valuation v of $\mathbf{t}$ *satisfies* $\mathcal{F}$ if for every node $\mathbf{n}$ of $\mathbf{t}$ and attribute a of the label of $\mathbf{n}$, $v(W(a(\mathbf{n}))) \in R^{a(\mathbf{n})}_{v(a(\mathbf{n}))}$. It follows immediately from the definitions that $\mathcal{F}(\mathbf{t})$ satisfies $\mathcal{F}$. Moreover, $\mathcal{F}(\mathbf{t})$ is the only valuation that satisfies $\mathcal{F}$. Indeed, suppose that v satisfies $\mathcal{F}$. An easy induction on l, using non-circularity, then shows that if $a(\mathbf{n})$ is defined in $\mathcal{F}_l(\mathbf{t})$ then $\mathcal{F}_l(\mathbf{t})(a(\mathbf{n})) = v(a(\mathbf{n}))$.

In MSO we just guess the values of the attributes, verify our guesses and select those nodes for which the result attribute is true. For ease of exposition we assume that all grammar symbols have all attributes, i.e., for every $X \in N \cup T$, $\mathrm{Inh}(X) \cup \mathrm{Syn}(X) = A$. We use set variables to represent the assignment of values: for each function $\alpha : A \to D$, Z_α will contain those nodes $\mathbf{n}$ such that for every attribute $a \in A$, $\mathcal{F}(\mathbf{t})(a(\mathbf{n})) = \alpha(a)$. We then only have to verify that all semantic rules are satisfied under this assignment. This can easily be done as it is well known that every regular language can be defined in MSO [45]. The result of the query expressed by $\mathcal{F}$ then consists of the nodes in all the Z_α where $\alpha(\mathit{result}) = 1$. Since for every tree there is only one assignment that satisfies $\mathcal{F}$ we can just existentially quantify over the Z_α.

We omit the formal construction of the MSO formula simulating $\mathcal{F}$ which is straightforward but tedious. ■

To prove the other direction, we show that extended AGs can compute the MSO-equivalence type of the input tree. Thereto, we introduce some terminology. For a node $\mathbf{n}$ of a tree $\mathbf{t}$, we write $(\mathbf{t}, \mathbf{n})$ to denote the finite structure $\mathbf{t}$ expanded with $\mathbf{n}$ as a distinguished constant. Let $\mathbf{t}_1$ and $\mathbf{t}_2$ be two trees, $\mathbf{n}_1$ a node of $\mathbf{t}_1$, $\mathbf{n}_2$ a node of $\mathbf{t}_2$ and k a natural number. We write $(\mathbf{t}_1, \mathbf{n}_1) \equiv_k (\mathbf{t}_2, \mathbf{n}_2)$ and say that $(\mathbf{t}_1, \mathbf{n}_1)$ and $(\mathbf{t}_2, \mathbf{n}_2)$ are $\equiv^{\mathrm{MSO}}_k$-*equivalent*, if for each MSO sentence φ of quantifier depth at most k,

$$(\mathbf{t}_1, \mathbf{n}_1) \models \varphi \Leftrightarrow (\mathbf{t}_2, \mathbf{n}_2) \models \varphi,$$

i.e., $(\mathbf{t}_1, \mathbf{n}_1)$ and $(\mathbf{t}_2, \mathbf{n}_2)$ cannot be distinguished by MSO sentences of quantifier depth (at most) k. It follows from the definition that $\equiv^{\mathrm{MSO}}_k$ is an equivalence relation. Moreover, $\equiv^{\mathrm{MSO}}_k$-equivalence can be nicely characterized by Ehrenfeucht games. The k-round MSO game on two structures $(\mathbf{t}_1, \mathbf{n}_1)$ and

$(\mathbf{t}_2, \mathbf{n}_2)$, denoted by $G_k^{\mathrm{MSO}}(\mathbf{t}_1, \mathbf{n}_1; \mathbf{t}_2, \mathbf{n}_2)$, is played by two players, the *spoiler* and the *duplicator*, in the following way. In each of the k rounds the spoiler decides whether he makes a *point move* or a *set move*. When the i-th move is a point move, he selects one element $\mathbf{p}_i \in \mathrm{Nodes}(\mathbf{t}_1)$ or $\mathbf{q}_i \in \mathrm{Nodes}(\mathbf{t}_2)$ and the duplicator answers by selecting one element of the other structure. When the i-th move is a set move, the spoiler chooses a set $\mathbf{P}_i \subseteq \mathrm{Nodes}(\mathbf{t}_1)$ or $\mathbf{Q}_i \subseteq \mathrm{Nodes}(\mathbf{t}_2)$ and the duplicator chooses a set in the other structure. After k rounds there are elements $\mathbf{p}_1, \ldots, \mathbf{p}_l$ and $\mathbf{q}_1, \ldots, \mathbf{q}_l$ that were chosen in the point moves in $\mathrm{Nodes}(\mathbf{t}_1)$ and $\mathrm{Nodes}(\mathbf{t}_2)$ respectively and there are sets $\mathbf{P}_1, \ldots, \mathbf{P}_n$ and $\mathbf{Q}_1, \ldots, \mathbf{Q}_n$ that were chosen in the set moves in $\mathrm{Nodes}(\mathbf{t}_1)$ and $\mathrm{Nodes}(\mathbf{t}_2)$, respectively. The duplicator wins this play if the mapping which maps $\mathbf{p}_i$ to $\mathbf{q}_i$ is a partial isomorphism from $(\mathbf{t}_1, \mathbf{n}_1, \mathbf{P}_1, \ldots, \mathbf{P}_n)$ to $(\mathbf{t}_2, \mathbf{n}_2, \mathbf{Q}_1, \ldots, \mathbf{Q}_n)$. That is, for all i and j, $\mathbf{p}_i \in P_j$ iff $\mathbf{q}_i \in Q_j$, and for every atomic formula $\varphi(\bar{x})$, $\mathbf{t} \models \varphi[\bar{\mathbf{p}}]$ iff $\mathbf{s} \models \varphi[\bar{\mathbf{q}}]$.

We say that the duplicator has a *winning strategy* in $G_k^{\mathrm{MSO}}(\mathbf{t}_1, \mathbf{n}_1; \mathbf{t}_2, \mathbf{n}_2)$, or shortly that he wins $G_k^{\mathrm{MSO}}(\mathbf{t}_1, \mathbf{n}_1; \mathbf{t}_2, \mathbf{n}_2)$, if he can win each play no matter which choices the spoiler makes.

See, e.g., the book by Ebbinghaus and Flum [19] for a proof of the next proposition.

Proposition 6.2 *The duplicator wins $G_k^{\mathrm{MSO}}(\mathbf{t}_1, \mathbf{n}_1; \mathbf{t}_2, \mathbf{n}_2)$ if and only if*

$$(\mathbf{t}_1, \mathbf{n}_1) \equiv_k^{\mathrm{MSO}} (\mathbf{t}_2, \mathbf{n}_2).$$

The relation $\equiv_k^{\mathrm{MSO}}$ has only a finite number of equivalence classes (see, e.g., [19]). We denote the set of these classes by Φ_k. We call the elements of Φ_k $\equiv_k^{\mathrm{MSO}}$-*equivalence types* (or just $\equiv_k^{\mathrm{MSO}}$-types). We denote by $\tau_k^{\mathrm{MSO}}(\mathbf{t}, \mathbf{n})$ the $\equiv_k^{\mathrm{MSO}}$-type of a tree $\mathbf{t}$ with a distinguished node $\mathbf{n}$; thus, $\tau_k^{\mathrm{MSO}}(\mathbf{t}, \mathbf{n})$ is the equivalence class of $(\mathbf{t}, \mathbf{n})$ w.r.t. $\equiv_k^{\mathrm{MSO}}$. By $\tau_k^{\mathrm{MSO}}(\mathbf{t})$ we denote the $\equiv_k^{\mathrm{MSO}}$-type of the tree $\mathbf{t}$ without a distinguished node. It is often useful to think of $\tau_k^{\mathrm{MSO}}(\mathbf{t}, \mathbf{n})$ as the set of MSO sentences of quantifier depth k that hold in $(\mathbf{t}, \mathbf{n})$. We abuse notation and sometimes write $\tau_k^{\mathrm{MSO}}(\mathbf{t}, \mathrm{root})$ for $\tau_k^{\mathrm{MSO}}(\mathbf{t}, \mathrm{root}(\mathbf{t}))$.

The next proposition contains the main ingredients of the proof of Theorem 6.4. Let $\varphi(x)$ be an MSO formula of quantifier-depth k. The first item of the next proposition says that $\mathbf{t} \models \varphi[\mathbf{n}]$ only depends on the $\equiv_k^{\mathrm{MSO}}$-type of the subtree rooted at $\mathbf{n}$, i.e., $\tau_k^{\mathrm{MSO}}(\mathbf{t}_\mathbf{n}, \mathbf{n})$, and on the $\equiv_k^{\mathrm{MSO}}$-type of the envelope of $\mathbf{t}$ at $\mathbf{n}$, i.e., $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_\mathbf{n}}, \mathbf{n})$. Hence, our original problem reduces to the computation of $\tau_k^{\mathrm{MSO}}(\mathbf{t}_\mathbf{n}, \mathbf{n})$ and $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_\mathbf{n}}, \mathbf{n})$ for each $\mathbf{n}$. The second item

(essentially) tells us that $\tau_k^{\mathrm{MSO}}(\mathbf{t_n}, \mathbf{n})$ can be computed in a bottom-up manner and from left to right within the siblings of each node. Finally, it follows (essentially) from the third item that $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t_n}}, \mathbf{n})$ can be computed in a top-down fashion once the $\equiv_k^{\mathrm{MSO}}$-types of all the $(\mathbf{t_m}, \mathbf{m})$ are known. The above two pass strategy, first compute the types of all subtrees and then compute the types of all envelopes, forms the core of the proof of Theorem 6.4.

Proposition 6.3 *Let k be a natural number, σ be a label, $\mathbf{t}$ and $\mathbf{s}$ be two trees, $\mathbf{n}$ be a node of $\mathbf{t}$ with children $\mathbf{n}_1, \ldots, \mathbf{n}_n$, and $\mathbf{m}$ be a node of $\mathbf{s}$ with children $\mathbf{m}_1, \ldots, \mathbf{m}_m$.*

1. *If $(\overline{\mathbf{t_n}}, \mathbf{n}) \equiv_k^{\mathrm{MSO}} (\overline{\mathbf{s_m}}, \mathbf{m})$ and $(\mathbf{t_n}, \mathbf{n}) \equiv_k^{\mathrm{MSO}} (\mathbf{s_m}, \mathbf{m})$, then $(\mathbf{t}, \mathbf{n}) \equiv_k^{\mathrm{MSO}} (\mathbf{s}, \mathbf{m})$.*

2. *If $(\sigma(\mathbf{t}_{\mathbf{n}_1}, \ldots, \mathbf{t}_{\mathbf{n}_{n-1}}), \mathrm{root}) \equiv_k^{\mathrm{MSO}} (\sigma(\mathbf{s}_{\mathbf{m}_1}, \ldots, \mathbf{s}_{\mathbf{m}_{m-1}}), \mathrm{root})$ and $(\mathbf{t}_{\mathbf{n}_n}, \mathbf{n}_n) \equiv_k^{\mathrm{MSO}} (\mathbf{s}_{\mathbf{m}_m}, \mathbf{m}_m)$, then $(\mathbf{t_n}, \mathbf{n}) \equiv_k^{\mathrm{MSO}} (\mathbf{s_m}, \mathbf{m})$.*

3. *Let the label of $\mathbf{n}$ and $\mathbf{m}$ be σ. For $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, if*

   - $(\overline{\mathbf{t_n}}, \mathbf{n}) \equiv_k^{\mathrm{MSO}} (\overline{\mathbf{s_m}}, \mathbf{m})$,
   - $(\sigma(\mathbf{t}_{\mathbf{n}_1}, \ldots, \mathbf{t}_{\mathbf{n}_{i-1}}), \mathrm{root}) \equiv_k^{\mathrm{MSO}} (\sigma(\mathbf{s}_{\mathbf{m}_1}, \ldots, \mathbf{s}_{\mathbf{m}_{j-1}}), \mathrm{root})$,
   - $(\sigma(\mathbf{t}_{\mathbf{n}_{i+1}}, \ldots, \mathbf{t}_{\mathbf{n}_n}), \mathrm{root}) \equiv_k^{\mathrm{MSO}} (\sigma(\mathbf{s}_{\mathbf{m}_{j+1}}, \ldots, \mathbf{s}_{\mathbf{m}_m}), \mathrm{root})$, *and*
   - *the label of $\mathbf{n}_i$ equals the label of $\mathbf{m}_j$,*

   then $(\overline{\mathbf{t}_{\mathbf{n}_i}}, \mathbf{n}_i) \equiv_k^{\mathrm{MSO}} (\overline{\mathbf{s}_{\mathbf{m}_j}}, \mathbf{m}_j)$.

Proof. Consider the first item. By Proposition 6.2, it suffices to show that the duplicator wins $G_k^{\mathrm{MSO}}(\mathbf{t}, \mathbf{n}; \mathbf{s}; \mathbf{m})$. We already know that he wins the subgames $G_k^{\mathrm{MSO}}(\mathbf{t_n}, \mathbf{n}; \mathbf{s_m}; \mathbf{m})$ and $G_k^{\mathrm{MSO}}(\overline{\mathbf{t_n}}, \mathbf{n}; \overline{\mathbf{s_m}}; \mathbf{m})$. The duplicator, therefore, combines these winning strategies as follows to obtain a winning strategy in $G_k^{\mathrm{MSO}}(\mathbf{t}, \mathbf{n}; \mathbf{s}; \mathbf{m})$. If the spoiler makes a point move then the duplicator answers corresponding to his winning strategy in the relevant subgame. If the spoiler makes a set move in, say, $\mathbf{t}$, choosing the sets $\mathbf{P}_1 \subseteq \mathrm{Nodes}(\mathbf{t_n})$ and $\mathbf{P}_2 \subseteq \mathrm{Nodes}(\overline{\mathbf{t_n}})$, then the duplicator responds with the set $\mathbf{Q}_1 \cup \mathbf{Q}_2$, where $\mathbf{Q}_1$ is the answer to $\mathbf{P}_1$ in the subgame $G_k^{\mathrm{MSO}}(\mathbf{t_n}, \mathbf{n}; \mathbf{s_m}; \mathbf{m})$ and $\mathbf{Q}_2$ is the answer to $\mathbf{P}_2$ in the subgame $G_k^{\mathrm{MSO}}(\overline{\mathbf{t_n}}, \mathbf{n}; \overline{\mathbf{s_m}}; \mathbf{m})$.

Note that this strategy is well-defined. Indeed, $(\mathbf{t_n}, \mathbf{n})$ and $(\overline{\mathbf{t_n}}, \mathbf{n})$ ($(\mathbf{s_m}, \mathbf{m})$ and $(\overline{\mathbf{s_m}}, \mathbf{m})$) have only $\mathbf{n}$ ($\mathbf{m}$) in common and due to the fact that $\mathbf{n}$ and $\mathbf{m}$

are distinguished constants in both subgames, the duplicator is forced to pick $\mathbf{n}$ whenever the spoiler picks $\mathbf{m}$, and vice versa. At the end of a game, the selected vertices define partial isomorphisms for the two pairs of respective substructures. As there is no relation in the vocabulary that can relate a node from, say, $\mathbf{t_n}$ to a node from $\overline{\mathbf{t_n}}$ (apart from $\mathbf{n}$), these mappings are also partial isomorphism for the whole structures. Hence, the above strategy is also winning.

We next focus on the third case and leave the second case to the reader. Here, there are altogether 4 subgames including the trivial game in which one structure consists only of $\mathbf{n}_i$ and the other of $\mathbf{m}_j$. The winning strategy in the game on $(\overline{\mathbf{t}_{\mathbf{n}_i}}, \mathbf{n}_i)$ and $(\overline{\mathbf{s}_{\mathbf{m}_j}}, \mathbf{m}_j)$ just combines the winning strategies in those 4 subgames (as explained for the first item). Again, at the end of a game, the selected vertices define partial isomorphisms for all pairs of respective substructures. To ensure that they also define a partial isomorphism between the entire structures one only has to check the preservation of the relations $<$ and E between the chosen elements, and the distinguished constants $\mathbf{n}_i$ and $\mathbf{m}_j$. This immediately follows from the following observations. The distinguished constants in the subgames make sure that (a) whenever in the game on $(\overline{\mathbf{t}_{\mathbf{n}_i}}, \mathbf{n}_i)$ and $(\overline{\mathbf{s}_{\mathbf{m}_j}}, \mathbf{m}_j)$ a child of $\mathbf{n}$ ($\mathbf{m}$) is chosen, the duplicator has to reply with a child of $\mathbf{m}$ ($\mathbf{n}$); and, (b) whenever $\mathbf{n}$ ($\mathbf{m}$) is chosen, the duplicator has to reply with $\mathbf{m}$ ($\mathbf{n}$). Additionally, the position of the subtrees in the whole tree make sure that $<$ is preserved w.r.t. $\mathbf{n}_i$ and $\mathbf{m}_j$. ■

We are ready to prove the main theorem of this section.

Theorem 6.4 *A query is expressible by an extended AG if and only if it is definable in MSO.*

Proof. The only-if direction was already given in Lemma 6.1.

Let $\varphi(x)$ be an MSO formula of quantifier depth k. We define an extended AG $\mathcal{F}$ expressing the query defined by φ. Define $D = \Phi_k \cup \{0, 1\}$ and $A = \{env, sub, result, \mathrm{lab}\}$, where *env* is inherited for all grammar symbols except for the start symbol for which it is synthesized, and *sub* and *result* are synthesized for all non-terminals and inherited for all terminals. The intended meaning is the following: for a node $\mathbf{n}$ of a tree $\mathbf{t}$,

- $\mathcal{F}(\mathbf{t})(sub(\mathbf{n})) = \tau_k^{\mathrm{MSO}}(\mathbf{t_n}, \mathbf{n})$,
- $\mathcal{F}(\mathbf{t})(env(\mathbf{n})) = \tau_k^{\mathrm{MSO}}(\overline{\mathbf{t_n}}, \mathbf{n})$, and

- $\mathcal{F}(\mathbf{t})(\mathit{result}(\mathbf{n})) = 1$ if and only if $\mathbf{t} \models \varphi[\mathbf{n}]$.

By Proposition 6.3(1), $\mathbf{t} \models \varphi[\mathbf{n}]$ only depends on $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_\mathbf{n}}, \mathbf{n})$ and $\tau_k^{\mathrm{MSO}}(\mathbf{t}_\mathbf{n}, \mathbf{n})$. Hence, $\mathcal{F}(\mathbf{t})(\mathit{result}(\mathbf{n}))$ only depends on the attribute values $\mathcal{F}(\mathbf{t})(\mathit{env}(\mathbf{n}))$ and $\mathcal{F}(\mathbf{t})(\mathit{sub}(\mathbf{n}))$.

As already hinted upon above, the extended AG we construct, works in two passes. In the first bottom-up pass all the *sub* attributes are computed (using the regular languages SUB, defined below); in the subsequent top-down pass all the *env* attributes are computed (using the regular languages ENV, defined below). Recall our convention that the start symbol cannot appear in the left-hand side of a production. Hence, whenever we encounter a node labeled with the start symbol, we know it is the root and can initiate our top-down pass.[5] During this second pass, there is enough information at each node $\mathbf{n}$ to decide whether $\mathbf{t} \models \varphi[\mathbf{n}]$.

We next define the regular languages SUB, which we use to compute $\equiv_k^{\mathrm{MSO}}$-types of subtrees in a bottom-up fashion. We again abbreviate $\tau_k^{\mathrm{MSO}}(\mathbf{t}, \mathrm{root}(\mathbf{t}))$ by $\tau_k^{\mathrm{MSO}}(\mathbf{t}, \mathrm{root})$. Define for $\theta \in \Phi_k$ and $X \in N$ the language $\mathrm{SUB}(X, \theta)$ over Φ_k as follows:

$$\theta_1 \cdots \theta_n \in \mathrm{SUB}(X, \theta)$$

if there exist trees $\mathbf{t}_1, \ldots, \mathbf{t}_n$ such that for $i = 1, \ldots, n$, $\tau_k^{\mathrm{MSO}}(\mathbf{t}_i, \mathrm{root}) = \theta_i$ and $\tau_k^{\mathrm{MSO}}(X(\mathbf{t}_1, \ldots, \mathbf{t}_n), \mathrm{root}) = \theta$. We show that $\mathrm{SUB}(X, \theta)$ is a regular language.

Lemma 6.5 *Let $X \in N$ and $\theta \in \Phi_k$. There exists a DFA $M = (S, \Phi_k, \delta, s_0, F)$ accepting $\mathrm{SUB}(X, \theta)$.*

Proof. Define $M = (S, \Phi_k, \delta, s_0, F)$ as the DFA where $S = \Phi_k \cup \{s_0\}$ and $F = \{\theta\}$. Define the transition function as follows: for all $\theta, \theta_1, \theta_2 \in \Phi_k$,

- $\delta(s_0, \theta) := \tau_k^{\mathrm{MSO}}(X(\mathbf{t}), \mathrm{root})$ with $\tau_k^{\mathrm{MSO}}(\mathbf{t}, \mathrm{root}) = \theta$, and
- $\delta(\theta_1, \theta) := \theta_2$, whenever there exists a tree $\mathbf{t}$ with an X-labeled node $\mathbf{n}$ of arity n (for some n) such that $\tau_k^{\mathrm{MSO}}(X(\mathbf{t}_{\mathbf{n}1}, \ldots, \mathbf{t}_{\mathbf{n}n-1}), \mathrm{root}) = \theta_1$, $\tau_k^{\mathrm{MSO}}(\mathbf{t}_{\mathbf{n}n}, \mathbf{n}n) = \theta$, and $\tau_k^{\mathrm{MSO}}(\mathbf{t}_\mathbf{n}, \mathbf{n}) = \theta_2$.

[5] As the sole purpose of this technical convention is to be able to identify the root of the input tree, it can easily be dispensed with by adding so-called root rules to the attribute grammar formalism (see, e.g., Giegerich for a definition of standard attribute grammars with root rules [21]).

By Proposition 6.3(2), it does not matter which trees in the equivalence classes θ, θ_1, and θ_2 we take. ■

Note that, $\delta^*(s_0, \theta_1 \cdots \theta_n) = \theta'$ if and only if there exist trees $\mathbf{t}_1, \ldots, \mathbf{t}_n$ such that for $i = 1, \ldots, n$, $\tau_k^{\mathrm{MSO}}(\mathbf{t}_i, \mathrm{root}) = \theta_i$ and $\tau_k^{\mathrm{MSO}}(X(\mathbf{t}_1, \ldots, \mathbf{t}_n), \mathrm{root}) = \theta'$.

Define for $\theta \in \Phi_k$, the language $\mathrm{ENV}(\theta)$ over $\Phi_k \cup \{\#\}$ as follows:

$$\bar{\theta} = \theta_0 \theta_1 \cdots \theta_{i-1} \# \theta_i \theta_{i+1} \cdots \theta_n \in \mathrm{ENV}(\theta)$$

iff

- $\theta_j \in \Phi_k$ for $j = 0, \ldots, n$, and
- there exists a tree $\mathbf{t}$ with a node $\mathbf{n}$ of arity n (for some n) such that $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_\mathbf{n}}, \mathbf{n}) = \theta_0$, $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_{\mathbf{n}i}}, \mathbf{n}i) = \theta$, and $\tau_k^{\mathrm{MSO}}(\mathbf{t}_{\mathbf{n}j}, \mathbf{n}j) = \theta_j$ for $j = 1, \ldots, n$.

By Proposition 6.3(3), $\bar{\theta} \in \mathrm{ENV}(\theta)$ only depends on $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_\mathbf{n}}, \mathbf{n})$, $\tau_k^{\mathrm{MSO}}(X(\mathbf{t}_{\mathbf{n}1}, \ldots, \mathbf{t}_{\mathbf{n}i-1}), \mathrm{root})$, $\tau_k^{\mathrm{MSO}}(X(\mathbf{t}_{\mathbf{n}i+1} \cdots \mathbf{t}_{\mathbf{n}n}), \mathrm{root})$, and the label of $\mathbf{n}i$ which in turn only depends on $\tau_k^{\mathrm{MSO}}(\mathbf{t}_{\mathbf{n}i}, \mathbf{n}i)$. In terms of the automaton M of Lemma 6.5, $\bar{\theta} \in \mathrm{ENV}(\theta)$ only depends on θ_0, $\delta^*(s_0, \theta_1 \cdots \theta_{i-1})$, $\delta^*(s_0, \theta_{i+1} \cdots \theta_n)$, and θ_i. It is, hence, not difficult to construct an automaton accepting $\mathrm{ENV}(\theta)$. Indeed, such an automaton stores θ_0 in its state; then simulates M until it reaches the symbol #; this gives the state $\delta^*(s_0, \theta_1 \cdots \theta_{i-1})$; hereafter M stores θ_i in its state and again simulates M until the end of the string which gives the state $\delta^*(s_0, \theta_{i+1} \cdots \theta_n)$; M then accepts if

$$\xi_X(\theta_0, \delta^*(s_0, \theta_1 \cdots \theta_{i-1}), \theta_i, \delta^*(s_0, \theta_{i+1} \cdots \theta_n)) = \theta.$$

Here, the function ξ_X is defined as follows. For θ_1, θ_2, θ_3, θ_4, $\theta \in \Phi_k$ and $X \in N$, $\xi_X(\theta_1, \theta_2, \theta_3, \theta_4) = \theta$ if there exists a tree $\mathbf{t}$ with an X-labeled node $\mathbf{n}$ of arity n (for some n) and an $i \in \{1, \ldots, n\}$, such that

- $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_\mathbf{n}}, \mathbf{n}) = \theta_1$;
- $\tau_k^{\mathrm{MSO}}(X(\mathbf{t}_{\mathbf{n}1} \ldots \mathbf{t}_{\mathbf{n}i-1}), \mathrm{root}) = \theta_2$;
- $\tau_k^{\mathrm{MSO}}(\mathbf{t}_{\mathbf{n}i}, \mathbf{n}i) = \theta_3$;
- $\tau_k^{\mathrm{MSO}}(X(\mathbf{t}_{\mathbf{n}i+1} \ldots \mathbf{t}_{\mathbf{n}n}), \mathrm{root}) = \theta_4$; and

- $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_{\mathbf{n}i}}, \mathbf{n}i) = \theta$.

By Proposition 6.3, for all $X \in N$ and $\theta_1, \theta_2 \in \Phi_k$, if $\theta_1 \neq \theta_2$ then $\mathrm{SUB}(X, \theta_1) \cap \mathrm{SUB}(X, \theta_2) = \emptyset$ and $\mathrm{ENV}(\theta_1) \cap \mathrm{ENV}(\theta_2) = \emptyset$. Also, for all $X \in N$, $\bigcup_{\theta \in \Phi_k} \mathrm{SUB}(X, \theta) = \Phi_k^*$ and $\bigcup_{\theta \in \Phi_k} \mathrm{ENV}(\theta) = \Phi_k^* \# \Phi_k^*$.

We are finally ready to define the semantic rules of $\mathcal{F}$. For every production $X \to r$, define in the context $(X \to r, sub, 0)$ the rule

$$sub(0) := \langle \sigma_0 = \varepsilon, \sigma_1 = sub, \ldots, \sigma_{|r|} = sub; \\ (R_\theta = \mathrm{SUB}(X, \theta))_{\theta \in \Phi_k}, R_0 = \{0,1\}^* \rangle.$$

The R_d's that are not mentioned are defined as the empty set. For every i such that $r(i) = \sigma$ is a terminal define in the context $(X \to r, sub, i)$ the rule

$$sub(i) := \langle \sigma_0 = \varepsilon, \sigma_1 = \varepsilon, \ldots, \sigma_{|r|} = \varepsilon; R_{\theta_\sigma} = \{\varepsilon\}, R_0 = DD^* \rangle.$$

The above rule just assigns the type $\theta_\sigma = \tau_k^{\mathrm{MSO}}(\mathbf{t}(\sigma), \mathrm{root})$ to every nonterminal σ. For $i = 1, \ldots, |r|$, define in the context $(X \to r, env, i)$ the rule

$$env(i) := \langle \sigma_0 = env, \sigma_1 = sub, \ldots, \sigma_{|r|} = sub; \\ (R_\theta = \mathrm{ENV}(\theta))_{\theta \in \Phi_k}, R_0 = \{0,1,\#\}^* \rangle.$$

For the start symbol, define in the context $(U \to r, env, 0)$ the rule

$$env(0) := \langle \sigma_0 = \varepsilon, \sigma_1 = \varepsilon, \ldots, \sigma_{|r|} = \varepsilon; R_{\theta(U)} = \{\varepsilon\}, R_0 = DD^* \rangle,$$

where $\theta(U) = \tau_k^{\mathrm{MSO}}\big(\overline{\mathbf{t}(U)}, \mathrm{root}(\mathbf{t}(U))\big)$. Finally, add in the context $(X \to r, result, 0)$ the rule

$$result(0) := \langle \sigma_0 = (env, sub), \sigma_1 = \varepsilon, \ldots, \sigma_{|r|} = \varepsilon; R_1, R_0 = D^* - R_1 \rangle,$$

and for every i such that $r(i)$ is a terminal, add in the context $(X \to r, result, i)$ the rule

$$result(i) := \langle \sigma_0 = \varepsilon, \sigma_1 = \varepsilon, \ldots, \sigma_{i-1} = \varepsilon, \sigma_i = (env, sub), \\ \sigma_{i+1} = \varepsilon, \ldots, \sigma_{|r|} = \varepsilon; R_1, R_0 = (D \cup \{\#\})^* - R_1 \rangle,$$

where R_1 consists of those two letter strings $\theta_1\theta_2 \in \Phi_k^2$ for which there exists a tree $\mathbf{t}$ with a node $\mathbf{n}$, with $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_{\mathbf{n}}}, \mathbf{n}) = \theta_1$, $\tau_k^{\mathrm{MSO}}(\mathbf{t}_{\mathbf{n}}, \mathrm{root}) = \theta_2$, and $\mathbf{t} \models \varphi[\mathbf{n}]$. ■

7 Optimization

An important research topic in the theory of query languages is that of optimization of queries. This comprises, for example, the detection and elimination of subqueries that always return the empty relation, or more general, the rewriting of queries, stated in a certain formalism, into equivalent ones that can be evaluated more efficiently. The central problem in the case of the latter is, hence, to decide whether the rewritten queries are indeed equivalent to the original ones. In this section we study the complexity of the emptiness and equivalence test of extended AGs. Interestingly, these results will be applied in the next section to obtain a new upper bound for deciding equivalence of Region Algebra expressions introduced by Consens and Milo [14].

We consider the following problems:

- **Non-emptiness**: Given an extended AG $\mathcal{F}$, does there exists a tree $\mathbf{t}$ and a node $\mathbf{n}$ of $\mathbf{t}$ such that $\mathcal{F}(\mathbf{t})(\mathit{result}(\mathbf{n})) = 1$?
- **Equivalence**: Given two extended AGs $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same grammar, do $\mathcal{F}_1$ and $\mathcal{F}_2$ express the same query?

To show the EXPTIME-hardness for the above decision problems we use a reduction from TWO PERSON CORRIDOR TILING which is known to be complete for EXPTIME [12]. The matching upper bound is obtained by a reduction to the emptiness problem of NBTAs which are defined in Section 2.4.

We start with the lower bound. For natural numbers n and m we view $\{1, \ldots, n\} \times \{1, \ldots, m\}$ as a rectangle consisting of m rows of width n. Let T be a finite set of tiles, let $H, V \subseteq T \times T$ be horizontal and vertical constraints, and let $\bar{b} = b_1, \ldots, b_n, \bar{t} = t_1, \ldots, t_n \in T^n$ be the bottom and the top row. A *corridor tiling* from $\bar{b}$ to $\bar{t}$ is a mapping $\lambda : \{1, \ldots, n\} \times \{1, \ldots, m\} \to T$, for some natural number m, such that

- the first row is $\bar{b}$, that is, $\lambda(1,1) = b_1, \ldots, \lambda(1,n) = b_n$;
- the m-th row is $\bar{t}$, that is, $\lambda(m,1) = t_1, \ldots, \lambda(m,n) = t_n$;
- for $i = 1, \ldots, n-1$ and $j = 1, \ldots, m$, $(\lambda(i,j), \lambda(i+1,j)) \in H$; and
- for $i = 1, \ldots, n$ and $j = 1, \ldots, m-1$, $(\lambda(i,j), \lambda(i,j+1)) \in V$.

In a two person corridor tiling game from $\bar{b}$ to $\bar{t}$, two players, on turn, place tiles row wise from bottom to top, and from left to right in each row. The first player starts and each newly placed tile should be consistent with the tiles already placed. The first player tries to make a corridor tiling from $\bar{b}$ to $\bar{t}$, whereas the second player tries to prevent this. If the first player always can achieve such a tiling no matter how the second player plays, then we say that player one *wins* the corridor game. A player that puts down a tile not consistent with the tiles already placed, immediately looses.

TWO PERSON CORRIDOR TILING is the problem to decide, given a set of tiles T, $H, V \subseteq T \times T$, a sequence of tiles $\bar{b} = b_1, \ldots, b_n$ and $\bar{t} = t_1, \ldots, t_n \in T^n$, whether player one wins the corridor game.

Lemma 7.1 *Deciding non-emptiness of extended AGs is hard for EXPTIME.*

Proof. The proof is a reduction from TWO PERSON CORRIDOR TILING to non-emptiness of extended AGs. A strategy for player one can be represented by a tree where the nodes are labeled with tiles. Indeed, if we put the rows of a tiling next to each other rather than on top of each other, then every branch, i.e., the sequence of labels from the root to a leaf, of a tree represents a possible tiling. If we forget about the start row $\bar{b}$ for a moment, then the odd depth nodes have no siblings and represent moves of player one and the even depth nodes do have siblings and represent all the choices of player two. A strategy is then winning when every branch is either a corridor tiling or is a tiling where player two made a false move.

The extended AG we construct will only accept trees, by selecting the root, that correspond to winning strategies for player one. The AG essentially only has to check the horizontal and vertical constraints. Since n, the width of the corridor, is constant, the vertical constraints can be checked by storing at each node the tile carried by its n-th ancestor. The horizontal constraints can be checked for each node by looking at the tile carried by its parent.

Let $(T, H, V, b_1, \ldots, b_n, t_1, \ldots, t_n)$ be an instance of TWO PERSON CORRIDOR TILING where $T = \{c_1, \ldots, c_k\}$. Define

$$G_{\text{corr}} = (N_{\text{corr}}, T_{\text{corr}}, P_{\text{corr}}, U_{\text{corr}})$$

as the ECFG, where the set of terminals T_{corr} contains only the symbol Ξ, and the set of non-terminals N_{corr} consists of all triples $\{(c, i, j) \mid c \in T, i \in \{1, 2\}, j \in \{1, \ldots, n\}\}$ together with the set $\{b_1, \ldots, b_n, t_1, \ldots, t_n\}$. If a node

is labeled with (c, i, j) then this means that player i has put tile c on the j-th square of the current row. The terminal Ξ functions as an end delimiter, indicating that either the end row $\bar{t}$ has been reached or that player two has put a tile on the board that is inconsistent with the tiles already present. The set of productions of G_{corr} now consists of the following rules:

1. $U_{\text{corr}} \to b_1$;
2. $b_n \to (c_1, 1, 1) + \cdots + (c_k, 1, 1) + t_1$ and $b_i \to b_{i+1}$ for $i = 1, \ldots, n-1$;
3. $t_n \to \Xi$ and $t_i \to t_{i+1}$ for $i = 1, \ldots, n-1$;
4. $$(c, 1, j) \to (c_1, 2, j+1) \cdots (c_k, 2, j+1),$$
   and
   $$(c, 2, j) \to (c_1, 1, j+1) + \cdots + (c_k, 1, j+1) + \Xi,$$
   for each $c \in P$ and $j \in \{1, \ldots, n-1\}$; and
5. $$(c, 1, n) \to (c_1, 2, 1) \cdots (c_k, 2, 1) + t_1,$$
   and
   $$(c, 2, n) \to (c_1, 1, 1) + \cdots + (c_k, 1, 1) + \Xi + t_1,$$
   for each $c \in P$.

If $\mathbf{t}$ is a derivation tree and $\mathbf{n}$ is a node of $\mathbf{t}$, then we call the j-th node on the path from the parent of $\mathbf{n}$ to the root, the j-th *ancestor* of $\mathbf{n}$. If $\mathbf{n}$ is labeled with (c, i, j) then we say that $\mathbf{n}$ is *admissible* if $(c', c) \in V$ where the n-th ancestor of $\mathbf{n}$ contains the tile c' in its label, and, additionally, if $j > 1$, then $(c'', c) \in H$ where $(c'', i'', j-1)$ is the label of the parent of $\mathbf{n}$.

The extended AG $\mathcal{F}$ has attributes $A = \{1, \ldots, n, \mathit{local}, \mathit{result}, \text{lab}\}$ and semantic domain $\{0, 1, c_1, \ldots, c_k\}$, and works in three passes. In the first top-down pass it defines the inherited attributes $1, \ldots, n$ such that for $j = 1, \ldots, n$, $\mathcal{F}(\mathbf{t})(j(\mathbf{n}))$ equals the tile in the label of the j-th ancestor of $\mathbf{n}$ for each node $\mathbf{n}$ of $\mathbf{t}$. Next, $\mathcal{F}$ uses these attributes to check local consistency of the tiling. More precisely, the attribute *local* is defined true for a node $\mathbf{n}$ labeled with $(c, 1, j)$ iff $\mathbf{n}$ is admissible; and the attribute *local* is defined true for a node $\mathbf{n}$ labeled with $(c, 2, j)$ iff $\mathbf{n}$ is admissible or $\mathbf{n}$ is not admissible and the only child of $\mathbf{n}$ is labeled with Ξ (the latter captures the case where player

one wins when player two uses a wrong tile). Finally, $\mathcal{F}$ checks whether all attributes *local* are true by making a bottom-up pass through the tree. If the latter is the case then $\mathcal{F}$ selects the root. Clearly, $\mathcal{F}$ can be constructed in polynomial time and is non-empty if and only if player one wins the corridor tiling game. We omit the formal description of $\mathcal{F}$. ■

Non-emptiness of extended AGs can in fact also be decided in EXPTIME. The proof essentially works as follows. For each extended AG $\mathcal{F}$ we construct an NBTA $T_{\mathcal{F}}$ guessing the attribute values at each node; it then accepts when the *result* attribute of at least one node is true. Since the size of $T_{\mathcal{F}}$ will be exponential in the size of $\mathcal{F}$ and non-emptiness of NBTAs can be checked in polynomial time (see Lemma 2.8), we obtain an EXPTIME algorithm for testing non-emptiness of extended AGs.

Bloem and Engelfriet [6] already showed that tree automata can guess attribute values of nodes defined by *standard* finite-valued attribute grammars on ranked trees. We must extend this technique to unranked trees and automata, and must control the sizes of the NFAs involved in the transition function of the automaton. In particular, we control these sizes by first describing the transition function by nondeterministic two-way automata with a pebble which can be transformed into equivalent one-way nondeterministic automata with only an exponential size increase.

Theorem 7.2 *Deciding non-emptiness of extended AGs is EXPTIME-complete.*

Proof. EXPTIME-hardness has just been shown in Lemma 7.1, so it remains to show that non-emptiness is in EXPTIME.

Let $\mathcal{F}$ be an extended AG over the grammar $G = (N, T, P, U)$. Recall that all regular languages R_d are represented by NFAs. W.l.o.g., we assume that every grammar symbol has all attributes, i.e., for all $X \in N \cup T$, $\mathrm{Inh}(X) \cup \mathrm{Syn}(X) = A$. As mentioned above, we construct an NBTA $T_{\mathcal{F}}$ such that $L(T_{\mathcal{F}}) \neq \emptyset$ if and only if $\mathcal{F}$ is non-empty. The size of $T_{\mathcal{F}}$ will be exponential in the size of $\mathcal{F}$. That is, the set of states of $T_{\mathcal{F}}$ and the NFAs representing transition functions will be exponential in the size of $\mathcal{F}$. By Lemma 2.8, non-emptiness of $T_{\mathcal{F}}$ can be checked in time exponential in the size of $\mathcal{F}$. Hence, the theorem follows.

The automaton $T_{\mathcal{F}}$ essentially guesses the values of the attributes and then verifies whether they satisfy all semantic rules. Therefore, we use as

states tuples (α, o, p, i) where $\alpha : A \to D$ is a function, $o \in \{0,1\}$, $p \in P \cup T$ and $i \in \{1, \ldots, s\}$, where $s = \max\{|r| \mid X \to r \in P\}$.

The intended meaning of the states is as follows. If a valid state assignment (cf. Section 2.4) of $T_{\mathcal{F}}$, assigns the state $q = (\alpha, o, p, i)$ to a node $\mathbf{n}$ of an input tree, then

- α represents the values of the attributes of $\mathbf{n}$; i.e., for all $a \in A$, $\mathcal{F}(\mathbf{t})(a(\mathbf{n})) = \alpha(a)$;
- $o = 1$ if and only if a node in the subtree rooted at $\mathbf{n}$ has been selected;
- if $\mathbf{n}$ is an internal node then $p \in P$ and $\mathbf{n}$ is derived by p; if $\mathbf{n}$ is a leaf then $p \in T$ and $\mathbf{n}$ is labeled by p; and
- if $\mathbf{n}$ is the root then $i = 1$; otherwise, if the parent $\mathbf{p}$ of $\mathbf{n}$ is derived by $p' \to r$, $\mathbf{n}$ is the j-th child of $\mathbf{p}$, and w is the string formed by the children of $\mathbf{p}$, then $\mathrm{pos}_r(j, w) = i$.

For a tuple $q = (\alpha, o, p, i) \in Q$, we denote o by $q.o$, α by $q.\alpha$, p by $q.p$, and i by $q.i$. If $q.p = X \to r \in P$ then we denote X by $q.X$ and r by $q.r$, and if $p \in T$ then we denote p also by $q.X$. If $\alpha : A \to D$ is a function and $\sigma = a_1 \cdots a_n$ is a sequence of attributes then we denote the string $\alpha(a_1) \cdots \alpha(a_n)$ by $\alpha(\sigma)$.

The set of final states F is defined as

$$\{q \in Q \mid q.o = 1,\ q.X = U \text{ and } q.i = 1\}.$$

We define the transition function. For all $\alpha \in A \to D$, $\sigma_1, \sigma_2 \in T$, $o \in \{0,1\}$, and $i \in \{1, \ldots, s\}$, define

$$\delta\big((\alpha, o, \sigma_1, i), \sigma_2\big) := \begin{cases} \{\varepsilon\} & \text{if } \sigma_1 = \sigma_2 \text{ and } o = \alpha(\mathit{result}) \ ; \\ \emptyset & \text{otherwise.} \end{cases}$$

For all $X \in N$ and $q \in Q$, if $q.X \neq X$ then $\delta(q, X) = \emptyset$; otherwise, if $q.X = X$ then $q_1 \cdots q_n \in \delta(q, X)$ iff

1. $q_1.X \cdots q_n.X \in L(q.r)$;
2. for all $j = 1, \ldots, n$, $\mathrm{pos}_{q.r}(j, q_1.X \cdots q_n.X) = q_j.i$;
3. for every synthesized attribute a of X, defined by the rule

$$\langle \sigma_0, \ldots, \sigma_{|q.r|}; (R_d)_{d \in D} \rangle,$$

we must have:

$$q.\alpha(\sigma_0) \cdot q_1.\alpha(\sigma_{q_1.i}) \cdots q_n.\alpha(\sigma_{q_n.i}) \in R_{q.\alpha(a)};$$

4. for all $j = 1, \ldots, n$, for every inherited attribute a of $q_j.X$, defined by the rule

$$\langle \sigma_0, \ldots, \sigma_{|q.r|}; (R_d)_{d \in D} \rangle,$$

we must have:

$$q.\alpha(\sigma_0) \cdot q_1.\alpha(\sigma_{q_1.i}) \cdots q_{j-1}.\alpha(\sigma_{q_{j-1}.i}) \# q_j.\alpha(\sigma_{q_j.i}) \\ q_{j+1}.\alpha(\sigma_{q_{j+1}.i}) \cdots q_n.\alpha(\sigma_{q_n.i}) \in R_{q.\alpha(a)};$$

and

5. $q.o = 1$ if and only if $q.\alpha(\mathit{result}) = 1$ or there exists a $j \in \{1, \ldots, n\}$ such that $q_j.o = 1$.

We show that conditions (1–5) are regular. Moreover, they can be defined by NFAs whose size is exponential in the size of $\mathcal{F}$. The result then follows since the size of the NFA computing the intersection of a constant number of NFAs is polynomial in the sizes of those NFAs.

- (1) and (2) are checked by the the NFA $M_{q.r}$ obtained from $q.r$ as described in Lemma 2.3 whose size is linear in r.
- For (3), we describe a two-way nondeterministic automaton M_1. By Proposition 2.9, M_1 can be transformed into an equivalent one-way NFA whose size is exponential in M_1. M_1 makes one pass through the input string for every synthesized attribute a of X simulating the NFA for $R_{q.\alpha(a)}$. If the latter accepts then M_1 walks back to the beginning of the input string and treats the next synthesized attribute or accepts if all synthesized attributes have been accounted for; M_1 rejects if the NFA for $R_{q.\alpha(a)}$ rejects. This needs only a linear number of states in the sizes of the NFAs representing transition functions and the set of attributes.

- For (4), we describe a two-way nondeterministic automaton M_2 with a pebble. By Proposition 2.9, M_2 can be transformed into an equivalent one-way NFA whose size is exponential in M_2. M_2 successively puts its pebble on each position of the input string. Suppose M_2 has just put the pebble on position j, then, for every inherited attribute a of $q_j.X$, M_2 walks back to the beginning of the input string and simulates the NFA for $R_{q_i.\alpha(a)}$, pretending to read # the moment it encounters the pebble. If the NFA for $R_{q_i.\alpha(a)}$ accepts, then M_2 walks back to the beginning of the input string and treats the next inherited attribute of $q_i.X$, or, if all inherited attributes of $q_i.X$ have been considered, moves the pebble to position $j+1$ and repeats the same procedure. If M_2 has put its pebble on all positions it accepts. This needs only a number of states linear in the size of the NFAs representing transition functions and the set of attributes.

- (5) can be done by making one pass over the input string using a constant number of states.

This concludes the proof of the theorem. ■

Let us now turn to the equivalence problem. This problem is actually polynomial-time equivalent to the complement of the non-emptiness problem (i.e., the emptiness problem), and hence it is also EXPTIME-complete. Indeed, $\mathcal{F}$ expresses the constant empty query if and only if it is equivalent to a trivial extended AG that expresses this query, and conversely, we can easily test if $\mathcal{F}_1$ and $\mathcal{F}_2$ express the same query by constructing an extended AG that first runs $\mathcal{F}_1$ and $\mathcal{F}_2$ independently, and then defines the value of *result* of a node to be 0 iff the values of *result* for $\mathcal{F}_1$ and $\mathcal{F}_2$ on that node agree. This gives the following theorem.

Theorem 7.3 *Deciding equivalence of extended AGs is EXPTIME-complete.*

8 Application: optimization of Region Algebra expressions

The region algebra introduced by Consens and Milo [14, 13] is a set-at-a-time algebra, based on the PAT algebra [42], for manipulating text regions. In this section we show that any Region Algebra expression can be simulated

by an extended AG of polynomial size. This then leads to an EXPTIME algorithm for the equivalence and emptiness test of Region Algebra expressions. The algorithm of Consens and Milo is based on the equivalence test for first-order logic formulas over trees which has a non-elementary lower bound and, therefore, conceals the real upper bound of the former problem. Our algorithm drastically improves the complexity of the equivalence test for the Region Algebra and matches more closely the coNP lower bound [14].

It should be pointed out that our definition differs slightly from the one in [14]. Indeed, we restrict ourselves to regular languages as patterns, while Consens and Milo do not use a particular pattern language. This is no loss of generality since

- on the one hand, regular languages are the most commonly used pattern language in the context of document databases; and,
- on the other hand, the huge complexity of the algorithm of [14] is not due to the pattern language at hand, but is due to quantifier alternation of the resulting first-order logic formula, induced by combinations of the operators '$-$' (difference) and $<$, $>$, $\subset$, and $\supset$.

Definition 8.1 A *region index schema* $\mathcal{I} = (S_1, \ldots, S_n, \Sigma)$ consists of a set of region names $S_1, \ldots, S_n$ and a finite alphabet Σ.

If N is a natural number, then a *region over* N is a pair (i, j) with $i \leq j$ and $i, j \in \{1, \ldots, N\}$.

An *instance* I of a region index schema $\mathcal{I}$ consists of a string $I(\omega) = a_1 \ldots a_{N_I} \in \Sigma^*$ with $N_I > 0$, and a mapping (also denoted by I) associating to each region name S a set of regions over N_I.

We abbreviate $r \in \bigcup_{i=1}^{n} I(S_i)$ by $r \in I$. We use the notation $L(r)$ (respectively $R(r)$) to denote the location of the left (respectively right) endpoint of a region r and denote by $\omega(r)$ the string $a_{L(r)} \ldots a_{R(r)}$.

Example 8.2 Consider the region index schema $\mathcal{I} = (\mathbf{Proc}, \mathbf{Func}, \mathbf{Var}, \Sigma)$. In Figure 6, an example of an instance over $\mathcal{I}$ is depicted. Here, $N_I = 16$, $I(\omega) = \text{abcdefghijklmnop}$, $I(\mathbf{Proc}) = \{(1, 16), (6, 10)\}$, $I(\mathbf{Func}) = \{(12, 16)\}$ and $I(\mathbf{Var}) = \{(2, 3), (6, 7), (12, 13)\}$. ■

For two regions r and s in I define:

- $r < s$ if $R(r) < L(s)$ (r *precedes* s); and

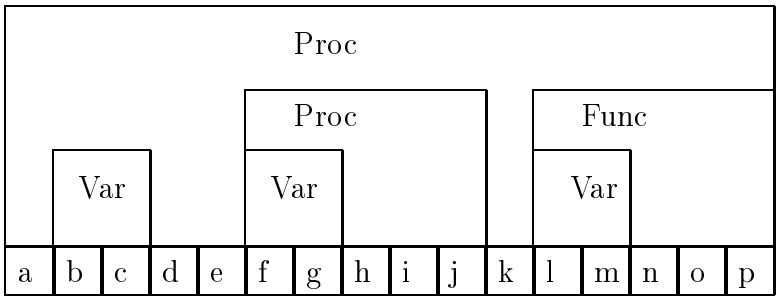

Figure 6: An instance I over the region index schema of Example 8.2

- $r \subset s$ if $L(s) < L(r)$ and $R(r) \leq R(s)$, or $L(s) \leq L(r)$ and $R(r) < R(s)$ (r is *included in* s).

We also allow the dual operators $r > s$ and $r \supset s$ which have the obvious meaning.

Definition 8.3 An instance I is *hierarchical* if

- $I(S) \cap I(S') = \emptyset$ for all region names S and S' in $\mathcal{I}$, and
- for all $r, s \in I$, one of the following holds: $r < s$, $s < r$, $r \subset s$ or $s \subset r$.

The last condition simply says that if two regions overlap then one is strictly contained in the other.

The instance in Figure 6 is hierarchical. Like in [14], we only consider hierarchical instances. We now define the Region Algebra.

Definition 8.4 *Region Algebra expressions over* $\mathcal{I} = (S_1, \ldots, S_n, \Sigma)$ are inductively defined as follows:

- every region name of $\mathcal{I}$ is a Region Algebra expression;
- if e_1 and e_2 are Region Algebra expressions then $e_1 \cup e_2$, $e_1 - e_2$, $e_1 \subset e_2$, $e_1 < e_2$, $e_1 \supset e_2$, and $e_1 > e_2$ are also Region Algebra expressions;
- if e is a Region Algebra expression and R is a regular language then $\sigma_R(e)$ is a Region Algebra expression.

The semantics of a Region Algebra expression on an instance I is defined as follows:

$$\begin{array}{lcl} [\![S]\!]^I & := & \{r \mid r \in I(S)\}; \\ [\![\sigma_R(e)]\!]^I & := & \{r \mid r \in [\![e]\!]^I \text{ and } \omega(r) \in R\}; \\ [\![e_1 \cup e_2]\!]^I & := & [\![e_1]\!]^I \cup [\![e_2]\!]^I; \\ [\![e_1 - e_2]\!]^I & := & [\![e_1]\!]^I - [\![e_2]\!]^I; \end{array}$$

and for $\star \in \{<, >, \subset, \supset\}$:

$$[\![e_1 \star e_2]\!]^I \;:=\; \{r \mid r \in [\![e_1]\!]^I \text{ and } \exists s \in [\![e_2]\!]^I \text{ such that } r \star s\}.$$

Example 8.5 The Region Algebra expression $\mathbf{Proc} \supset \sigma_{\Sigma^* \mathrm{start} \Sigma^*}(\mathbf{Proc})$ defines all the **Proc** regions which contain a **Proc** region that contains the string start. ■

An important observation is that for any region index schema $\mathcal{I} = (S_1, \ldots, S_n, \Sigma)$ there exists an ECFG $G_{\mathcal{I}}$ such that any hierarchical instance of $\mathcal{I}$ 'corresponds' to a derivation tree of $G_{\mathcal{I}}$. This ECFG is defined as follows: $G_{\mathcal{I}} = (N, T, P, U)$, with $N = \{S_1, \ldots, S_n\}$, $T = \Sigma$, and where P consists of the rules

$$\begin{array}{lcll} p_0 & := & U & \rightarrow \quad (S_1 + \ldots + S_n + \Sigma)^+; \\ p_1 & := & S_1 & \rightarrow \quad (S_1 + \ldots + S_n + \Sigma)^+; \\ & & \vdots & \\ p_n & := & S_n & \rightarrow \quad (S_1 + \ldots + S_n + \Sigma)^+. \end{array}$$

For example, the derivation tree $\mathbf{t}_I$ of $G_{\mathcal{I}}$ representing the instance I of Figure 6 is depicted in Figure 7. Regions in I then correspond to nodes in $\mathbf{t}_I$ in the obvious way. We denote the node in $\mathbf{t}_I$ that corresponds to the region r by $\mathbf{n}_r$.

Since extended AGs can store results of subcomputations in their attributes, they are naturally closed under composition. It is, hence, no surprise that the translation of Region Algebra expressions into extended AGs proceeds by induction on the structure of the former.

Lemma 8.6 *For every Region Algebra expression e over $\mathcal{I}$ there exists an extended AG $\mathcal{F}_e$ over $G_{\mathcal{I}}$ such that for every hierarchical instance I and region $r \in I$, $r \in [\![e]\!]^I$ if and only if $\mathcal{F}_e(\mathbf{t}_I)(result_e(\mathbf{n}_r)) = 1$. Moreover, $\mathcal{F}_e$ can be constructed in time polynomial in the size of e.*

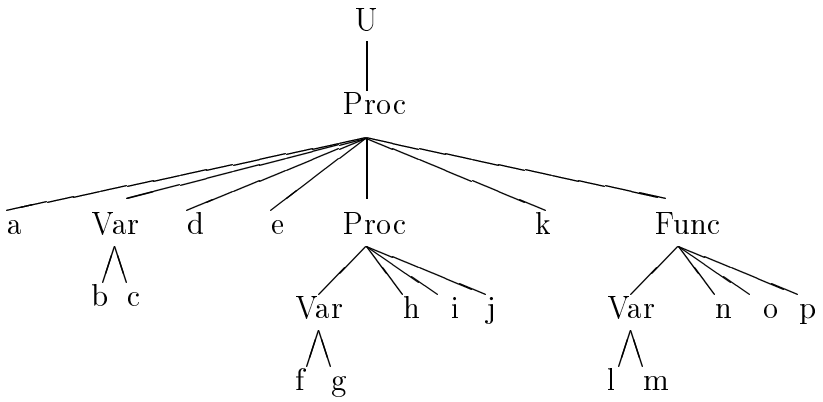

Figure 7: The tree $\mathbf{t}_I$ corresponding to the instance I of Figure 6.

Proof. The proof proceeds by induction on the structure of Region Algebra expressions. We represent the regular languages occurring as patterns in Region Algebra expressions by DFAs. The extended AG $\mathcal{F}_e$ will always contain the attribute $result_e$ which is synthesized for all region names. As before the R_d's that are not specified are assumed to be empty. Region Algebra expressions can only select regions, therefore, no attributes are defined for terminals.

1. $e = S_j$: $A_e = \{result_e, lab\}$; $D_e = \{0, 1, S_1, \ldots, S_n\} \cup \Sigma$; for $i = 1, \ldots, n$, define in the context $(p_i, result_e, 0)$ the rule

   $$result_e(0) := \langle \sigma_0 = \text{lab}, \sigma_1 = \varepsilon, \ldots, \sigma_{n+1} = \varepsilon; R_1 = \{S_j\}, R_0 = D_e^* - R_1 \rangle.$$

2. $e = \sigma_R(e_1)$: Let $M = (S, \Sigma, \delta, s_0, F)$ be the DFA accepting R with $S = \{s_0, \ldots, s_m\}$. Define $A_e = A_{e_1} \cup S$ and $D_e = D_{e_1} \cup S$. W.l.o.g, we assume $S \cap A_{e_1} = \emptyset$ and $S \cap D_{e_1} = \emptyset$. We define the semantic rules of $\mathcal{F}_e$ as the semantic rules of $\mathcal{F}_{e_1}$ extended with the ones we describe next.

   Each non-terminal has the synthesized attributes $s_0, \ldots, s_m$. They are defined in $\mathcal{F}_e$ such that for a region instance I and region $r \in I$, $\mathcal{F}_e(\mathbf{t}_I)(s(\mathbf{n}_r)) = s'$ if and only if $\delta^*(s, \omega(r)) = s'$. So, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, define in the context $(p_i, s_j, 0)$ the rule

   $$s_j(0) := \langle \sigma_0 = \varepsilon, \sigma_1 = (s_0, \ldots, s_m), \ldots, \sigma_n = (s_0, \ldots, s_m), \sigma_{n+1} = \text{lab}; (R_s^j)_{s \in S} \rangle.$$

It remains to define the regular languages $(R_s^j)_{s\in S}$. Note that the input strings for each R_s^j are of the form $\bar{w} = w_1 \cdots w_k$, where for $l = 1, \ldots, k$, $w_l \in S^{m+1}$ or $w_l \in \Sigma^*$. The DFA $M_{s,j}$ accepting R_s^j then works as follows: it starts in state s_j, if $w_1 \in S^{m+1}$ then $M_{s,j}$ continues in state s' where s' occurs on the $(j+1)$-th position of w_1 (this is the value of the attribute s_j); otherwise; if $w_1 \in \Sigma^*$ then $M_{s,j}$ continues in state $\delta^*(s_j, w_1)$. Formally, $M_{s,j}$ accepts $\bar{w}$ if there exists $j_1, \ldots, j_k \in \{0, \ldots, m\}$ such that

- if $w_1 \in S^{m+1}$ then $s_{j_1} = w_1(j+1)$;[6] if $w_1 \in \Sigma^*$ then $s_{j_1} = \delta^*(s_j, w_1)$;
- for $l = 2, \ldots, k$, if $w_l \in S^{m+1}$ then $s_{j_l} = w_l(j_{l-1}+1)$; if $w_l \in \Sigma^*$ then $s_{i_l} = \delta^*(s_{j_{l-1}}, w_l)$; and
- $s_{j_k} = s$.

Clearly, $M_{s,j}$ can be defined using a number of states polynomial in the size of S. The attribute $result_e$ then becomes true for a node $\mathbf{n}$, when $\mathcal{F}_e(\mathbf{t}_I)(s_0(\mathbf{n})) \in F$ and $\mathcal{F}_e(\mathbf{t}_I)(result_{e_1}(\mathbf{n})) = 1$. So, for $i = 1, \ldots, n$, define in the context $(p_i, result_e, 0)$ the rule

$$result_e(0) := \langle \sigma_0 = (s_0, result_{e_1}), \sigma_1 = \varepsilon, \ldots, \sigma_{n+1} = \varepsilon; \\ R_1 = \{s1 \mid s \in F\}, R_0 = D_e^* - R_1 \rangle.$$

In the following e will always depend on subexpressions e_1 and e_2. Hence, $\mathcal{F}_e$ always consist of $\mathcal{F}_{e_1}$ and $\mathcal{F}_{e_2}$ extended with rules for the new attributes. We, therefore, only specify the new rules. We will always assume that (apart from the attribute lab) A_{e_1}, A_{e_2} and the set of new attributes are disjoint.

3. $e = e_1 \cup e_2$: A node $\mathbf{n}$ is selected when

$$\mathcal{F}_e(\mathbf{t}_I)(result_{e_1}(\mathbf{n})) = 1 \text{ or } \mathcal{F}_e(\mathbf{t}_I)(result_{e_2}(\mathbf{n})) = 1.$$

Define $A_e = A_{e_1} \cup A_{e_2} \cup \{result_e\}$ and $D_e = D_{e_1} \cup D_{e_2}$. So, for $i = 1, \ldots, n$, define in the context $(p_i, result_e, 0)$ the rule

$$result_e(0) := \langle \sigma_0 = (result_{e_1}, result_{e_2}), \sigma_1 = \varepsilon, \ldots, \sigma_{n+1} = \varepsilon; \\ R_1 = \{01, 10\}, R_0 = D_e^* - R_1 \rangle.$$

[6] By $w_1(j+1)$, we denote the $(j+1)$-th position of the string w_1.

4. $e = e_1 - e_2$: A node $\mathbf{n}$ is selected when

$$\mathcal{F}_e(\mathbf{t}_I)(result_{e_1}(\mathbf{n})) = 1 \text{ and } \mathcal{F}_e(\mathbf{t}_I)(result_{e_2}(\mathbf{n})) = 0.$$

Define $A_e = A_{e_1} \cup A_{e_2} \cup \{result_e\}$ and $D_e = D_{e_1} \cup D_{e_2}$. So, for $i = 1, \ldots, n$, define in the context $(p_i, result_e, 0)$ the rule

$$result_e(0) := \langle \sigma_0 = (result_{e_1}, result_{e_2}), \sigma_1 = \varepsilon, \ldots, \sigma_{n+1} = \varepsilon; \\ R_1 = \{10\}, R_0 = D_e^* - R_1 \rangle.$$

5. $e = e_1 < e_2$: Define $A_e = A_{e_1} \cup A_{e_2} \cup \{right, result_e\}$ and $D_e = D_{e_1} \cup D_{e_2}$. Each non-terminal has the inherited attribute *right* such that for a region instance I and a region r, $\mathcal{F}_e(\mathbf{t}_I)(right(\mathbf{n}_r)) = 1$ if there exists a region s such that $r < s$ and $s \in [\![e_2]\!]^I$. Thus, for $j = 1, \ldots, n$, define in the context $(p_0, right, j)$ the rule

$$right(j) := \langle \sigma_0 = \varepsilon, \sigma_1 = result_{e_2}, \ldots, \sigma_n = result_{e_2}, \sigma_{n+1} = \varepsilon; \\ R_1, R_0 = (D_e \cup \{\#\})^* - R_1 \rangle,$$

where R_1 is the regular language that contains a string $w_1 \# a w_2$, with $w_1, w_2 \in \{0,1\}^*$ and $a \in \{0,1\}$, if w_2 contains a 1. For $i = 1, \ldots, n$ and $j = 1, \ldots, n$, define in the context $(p_i, right, j)$ the rule

$$right(j) := \langle \sigma_0 = right, \sigma_1 = result_{e_2}, \ldots, \sigma_n = result_{e_2}, \sigma_{n+1} = \varepsilon; \\ R_1, R_0 = (D_e \cup \{\#\})^* - R_1 \rangle,$$

where R_1 is the regular language that contains a string $a w_1 \# b w_2$, with $w_1, w_2 \in \{0,1\}^*$ and $a, b \in \{0,1\}$, if w_2 contains a 1 or $a = 1$. A node $\mathbf{n}$ is then selected when $\mathcal{F}_e(\mathbf{t}_I)(result_{e_1}(\mathbf{n})) = 1$ and $\mathcal{F}_e(\mathbf{t}_I)(right(\mathbf{n})) = 1$. So, for $i = 1, \ldots, n$, define in the context $(p_i, result_e, 0)$ the rule

$$result_e(0) := \langle \sigma_0 = (result_{e_1}, right), \sigma_1 = \varepsilon, \ldots, \sigma_{n+1} = \varepsilon; \\ R_1 = \{11\}, R_0 = D_e^* - R_1 \rangle.$$

6. $e = e_1 > e_2$: Similar to the previous case;

7. $e = e_1 \supset e_2$: Define $A_e = A_{e_1} \cup A_{e_2} \cup \{down, result_e\}$ and $D_e = D_{e_1} \cup D_{e_2}$. Each region name has the synthesized attribute *down* such that for a region instance I and a region r, $\mathcal{F}_e(\mathbf{t}_I)(down(\mathbf{n}_r)) = 1$ if there exists

a region s such that $r \supset s$ and $s \in [\![e_2]\!]^I$. So, for $i = 1, \ldots, n$, define in the context $(p_i, down, 0)$ the rule

$$down(0) := \langle \sigma_0 = \varepsilon, \sigma_1 = (result_{e_2}, down), \ldots, \sigma_n = (result_{e_2}, down), \sigma_{n+1} = \varepsilon; R_1, R_0 = D_e^* - R_1 \rangle,$$

where R_1 is the regular language that contains all strings containing at least one 1. A node $\mathbf{n}$ is then selected when

$$\mathcal{F}_e(\mathbf{t}_I)(result_{e_1}(\mathbf{n})) = 1$$

and

$$\mathcal{F}_e(\mathbf{t}_I)(down(\mathbf{n})) = 1.$$

So, for $j = 1, \ldots, n$, define in the context $(p_i, result_e, 0)$ the rule

$$result_e(0) := \langle \sigma_0 = (result_{e_1}, down), \sigma_1 = \varepsilon, \ldots, \sigma_{n+1} = \varepsilon; R_1 = \{11\}, R_0 = D_e^* - R_1 \rangle.$$

8. $e = e_1 \subset e_2$: Define $A_e = A_{e_1} \cup A_{e_2} \cup \{up, result_e\}$ and $D_e = D_{e_1} \cup D_{e_2}$. Each region name has the inherited attribute up such that for a region instance I and a region r, $\mathcal{F}_e(\mathbf{t}_I)(up(\mathbf{n}_r)) = 1$ if there exists a region s such that $r \subset s$ and $s \in [\![e_2]\!]^I$. So, for $j = 1, \ldots, n$, define in the context (p_0, up, j) the rule

$$up(j) := \langle \sigma_0 = \varepsilon, \sigma_1 = \varepsilon, \ldots, \sigma_{n+1} = \varepsilon; R_1 = \emptyset, R_0 = (D_e \cup \{\#\})^* \rangle.$$

For $i = 1, \ldots, n$ and $j = 1, \ldots, n$, define in the context (p_i, up, j) the rule

$$up(j) := \langle \sigma_0 = (up, result_{e_2}), \sigma_1 = \varepsilon, \ldots, \sigma_{n+1} = \varepsilon; R_1 = \{11\#, 10\#, 01\#\}, R_0 = (D_e \cup \{\#\})^* - R_1 \rangle.$$

A node $\mathbf{n}$ is then selected when $\mathcal{F}_e(\mathbf{t}_I)(result_{e_1}(\mathbf{n})) = 1$ and

$$\mathcal{F}_e(\mathbf{t}_I)(up(\mathbf{n})) = 1.$$

So, for $i = 1, \ldots, n$, define in the context $(p_i, result_e, 0)$ the rule

$$result_e(0) := \langle \sigma_0 = (result_{e_1}, up), \sigma_1 = \varepsilon, \ldots, \sigma_{n+1} = \varepsilon; R_1 = \{11\}, R_0 = D_e^* - R_1 \rangle.$$

■

We need the following definition to state the main result of this section.

Definition 8.7 A Region Algebra expression e over $\mathcal{I}$ is *empty* if for every hierarchical instance I over $\mathcal{I}$, $[\![e]\!]^I = \emptyset$. Two Region Algebra expressions e_1 and e_2 over $\mathcal{I}$ are *equivalent* if for every hierarchical instance I over $\mathcal{I}$, $[\![e_1]\!]^I = [\![e_2]\!]^I$.

Theorem 8.8 *Testing emptiness and equivalence of Region Algebra expressions is in* EXPTIME.

Proof. Although every hierarchical instance of $\mathcal{I} = (S_1, \ldots, S_n, \Sigma)$ can be represented as a derivation tree of $G_{\mathcal{I}}$, not every derivation tree of $G_{\mathcal{I}}$ is an hierarchical instance. Indeed, if an internal node has no siblings then it represents the same region as it parent. For example, the instance corresponding to the derivation tree

$$\begin{array}{c} U \\ \downarrow \\ \textbf{Proc} \\ \downarrow \\ \textbf{Func} \\ \downarrow \\ a \end{array}$$

is not hierarchical because **Proc** and **Func** represent the same region. An extended AG can easily check this condition by making one bottom-up pass through the tree. Another top-down pass then informs all nodes in the tree whether the tree represents an hierarchical instance.

If e is a Region Algebra expression, then we define $\mathcal{F}(e)$ as the extended AG $\mathcal{F}_e$, given by Lemma 8.6, that first checks whether the input tree is an hierarchical instance and if so, simulates e; otherwise it assigns false to the result attribute of any node. Hence, $\mathcal{F}(e)$ is empty if and only if e is empty. Further, if e_1 and e_2 are Region Algebra expressions, then, obviously, $\mathcal{F}(e_1)$ and $\mathcal{F}(e_2)$ are equivalent if and only if e_1 and e_2 are equivalent. Hence, the result follows by Lemma 7.2.

We describe the construction of $\mathcal{F}(e)$ in more detail. Define $A = A_e \cup \{subhier, hier, result\}$ and $D = D_e$, where A_e and D_e are the attribute set and the semantic domain of $\mathcal{F}_e$, respectively. The semantic rules of $\mathcal{F}(e)$ consists of those of $\mathcal{F}_e$ extended with the rules defining *subhier*, *hier*, and *result*.

Each region name has the synthesized attribute *subhier* such that for a region instance I and a region r, $\mathcal{F}_e(\mathbf{t}_I)(subhier(\mathbf{n}_r)) = 1$ if $\mathbf{t}_{\mathbf{n}_r}$ represents an hierarchical instance. So, for $i = 1, \ldots, n$, define in the context $(p_i, subhier, 0)$ the rule

$$subhier(0) := \langle \sigma_0 = \varepsilon, \sigma_1 = (\text{lab}, subhier), \ldots, \sigma_n = (\text{lab}, subhier), \\ \sigma_{n+1} = \text{lab}; R_0, R_1 = D^* - R_0, \rangle,$$

where R_0 is the regular language consisting of all strings containing at least one 0 and all the strings $\{S_1 1, \ldots, S_n 1\}$. This rule is correct, since $\mathbf{t}_{\mathbf{n}_r}$ does not represent an hierarchical instance only when at least one of its subtrees does not represent an hierarchical instance, or when $\mathbf{n}_r$ has just one child that, additionally, is labeled with a region name.

Each region name has the inherited attribute *hier* such that for a region instance I and a region r, $\mathcal{F}_e(\mathbf{t}_I)(hier(\mathbf{n}_r)) = 1$ if $\mathbf{t}_I$ represents an hierarchical instance. So, for $j = 1, \ldots, n$, define in the context $(p_0, hier, j)$ the rule

$$hier(j) := \langle \sigma_0 = \varepsilon, \sigma_1 = subhier, \ldots, \sigma_n = subhier, \sigma_{n+1} = \varepsilon; \\ R_1 = 1^*, R_0 = (D \cup \{\#\})^* - R_1 \rangle.$$

For $i = 1, \ldots, n$ and $j = 1, \ldots, n$, define in the context $(p_i, hier, j)$ the rule

$$hier(j) := \langle \sigma_0 = hier, \sigma_1 = \varepsilon, \ldots, \sigma_{n+1} = \varepsilon; \\ R_1 = \{1\#\}, R_0 = (D \cup \{\#\})^* - R_1 \rangle.$$

A node $\mathbf{n}$ is then selected when $\mathcal{F}_e(\mathbf{t}_I)(result_e(\mathbf{n})) = 1$ and $\mathcal{F}_e(\mathbf{t}_I)(hier(\mathbf{n})) = 1$. So, for $j = 1, \ldots, n$, define in the context $(p_i, result, 0)$ the rule

$$result(0) := \langle \sigma_0 = (result_e, hier), \sigma_1 = \varepsilon, \ldots, \sigma_n = \varepsilon; \\ R_1 = \{11\}, R_0 = D^* - R_1 \rangle.$$

■

9 Relational extended AGs

In this section we define relational extended AGs which can be viewed as extensions of relational BAGs [34]. The main difference with the extended AGs studied before (to which we refer by functional extended AGs) is that we

now associate *one* regular language R with each production, rather than with each position in a production and for each attribute. Specifically, we show that relational extended AGs express the same class of queries as extended AGs.

An attribute grammar vocabulary is now just a tuple (D, A, Att), where D is a finite semantic domain, A is a finite set of attributes, and Att is a function from $N \cup T$ to the powerset of A assigning to each grammar symbol a set of attributes. A *relational extended AG* $\mathcal{F}$ now associates to each production $p = X \to r$ a semantic rule $\langle \sigma_0, \ldots, \sigma_{|r|}; R_p \rangle$, where for $i \in \{0, \ldots, |r|\}$, σ_i is a sequence of attributes of $p(i)$ and R_p is a regular language over D. Let $\mathbf{t}$ be a derivation tree, $\mathbf{n}$ a node of $\mathbf{t}$ with children $\mathbf{n}_1, \ldots, \mathbf{n}_m$ derived by p, and let for $l \in \{1, \ldots, m\}$, $j_l = \mathrm{pos}_r(l, w)$, where w is the string formed by the labels of the children of $\mathbf{n}$. Then define $W(\mathbf{n})$ as the sequence $\sigma_0(\mathbf{n}) \cdot \sigma_{j_1}(\mathbf{n}_1) \cdots \sigma_{j_m}(\mathbf{n}_m)$. A *valuation of* $\mathbf{t}$ is again a function that maps each pair $(\mathbf{n}, a)$, where $\mathbf{n}$ is a node in $\mathbf{t}$ and a is an attribute of the label of $\mathbf{n}$, to an element of D, and that maps for every $\mathbf{n}$, $v((lab, \mathbf{n}))$ to the label of $\mathbf{n}$. A valuation v of $\mathbf{t}$ is said to *satisfy* $\mathcal{F}$ if $v(W(\mathbf{n})) \in R_p$ for every $p \in P$ and every internal node $\mathbf{n}$ derived by p.

A relational extended AG $\mathcal{F}$ can express queries in various ways. We consider two natural ones.

Definition 9.1 (i) A query $\mathcal{Q}$ is expressed *existentially* by a relational extended AG $\mathcal{F}$ if for every $\mathbf{t}$

$$\mathcal{Q}(\mathbf{t}) = \{\mathbf{n} \mid \text{there exists a valuation } v \text{ that satisfies } \mathcal{F} \text{ such that } v(\mathit{result}(\mathbf{n})) = 1\};$$

(ii) A query $\mathcal{Q}$ is expressed *universally* by a relational extended AG $\mathcal{F}$ if for every $\mathbf{t}$

$$\mathcal{Q}(\mathbf{t}) = \{\mathbf{n} \mid \text{for every valuation } v \text{ that satisfies } \mathcal{F},\ v(\mathit{result}(\mathbf{n})) = 1\}.$$

We give an example of the just introduced notions.

Example 9.2 Consider the ECFG of Example 4.5. Let $\mathcal{Q}$ be the query that selects every other poem. The following relational extended AG expresses $\mathcal{Q}$ existentially and universally. If p is the production DB $\to$ Poem$^+$, then define its associated rule as

$$\langle \sigma_0 = \varepsilon, \sigma_1 = \mathit{result}; R_p = (10)^*(1 + \varepsilon) \rangle,$$

here $A = \{result, \mathrm{lab}\}$, $D = \{0, 1\}$ and *result* is an attribute of Poem.

Note that this query $\mathcal{Q}$ is also expressed by a functional extended AG with $A = \{result, \mathrm{lab}\}$, $D = \{0, 1, \mathrm{Poem}, \mathrm{DB}\}$, $\mathrm{Inh}(\mathrm{Poem}) = \{result\}$, and $\mathrm{Syn}(\mathrm{Poem}) = \emptyset$. Define in the context $(p, result, 1)$ the semantic rule

$$result(1) := \langle \sigma_0 = \varepsilon, \sigma_1 = lab; R_1 = (\mathrm{PoemPoem})^* \# \mathrm{Poem}^* \rangle.$$

■

Functional and relational extended AGs are equally expressive as is shown in the next theorem. We just show that every query expressed existentially or universally by a relational extended AG can be defined in MSO and that every MSO definable query can be expressed both by an existential and a universal AG. The result then follows from Theorem 6.4.

Theorem 9.3 *The class of queries expressed existentially (universally) by relational extended AGs coincides with the class of queries expressed by extended AGs.*

Proof. The semantics of a relational extended AG can readily be defined in MSO. For ease of exposition we assume that all grammar symbols have all attributes, i.e., for every $X \in N \cup T$, $\mathrm{Inh}(X) \cup \mathrm{Syn}(X) = A$. We again use set variables Z_α, with α a function from A to D, to represent assignments of values to attributes. As in the proof of Lemma 6.1, we can construct an MSO formula $\psi((Z_\alpha)_{\alpha \in A \to D})$ such that whenever $\mathbf{t} \models \psi((Z_\alpha)_{\alpha \in A \to D})$ then

- the sets $(Z_\alpha)_{\alpha \in A \to D}$ are pairwise disjoint,
- $\bigcup_\alpha Z_\alpha = \mathrm{Nodes}(\mathbf{t})$, and
- the valuation v defined as, $v(a(\mathbf{n})) = \alpha(a)$ with $\mathbf{n} \in Z_\alpha$, satisfies $\mathcal{F}$.

The formula ψ just verifies the semantic rules of $\mathcal{F}$; and, since these are regular languages, this can be easily done in MSO. The following formulas then define the query expressed existentially respectively universally by $\mathcal{F}$

$$(\exists Z_\alpha)_{\alpha \in A \to D} \left(\psi((Z_\alpha)_{\alpha \in A \to D}) \wedge \bigvee \{ Z_\alpha(x) \mid \alpha(result) = 1 \} \right),$$

$$(\forall Z_\alpha)_{\alpha \in A \to D} \left(\psi((Z_\alpha)_{\alpha \in A \to D}) \to \bigvee \{ Z_\alpha(x) \mid \alpha(result) = 1 \} \right).$$

By Theorem 6.4, these MSO formulas can be transformed into an equivalent extended AG.

For the other direction, by Theorem 6.4, it suffices to show that any MSO definable query can be expressed by a relational extended AG under both the existential and the universal semantics. Let $\varphi(x)$ be an MSO formula of quantifier depth k. We now define a relational extended AG $\mathcal{F}$ that expresses φ under both the existential and the universal semantics. This relational extended AG just computes the $\equiv_k^{\mathrm{MSO}}$-type for every node of the input tree. We write $\tau_k^{\mathrm{MSO}}(\mathbf{t}, \mathrm{root})$ for $\tau_k^{\mathrm{MSO}}(\mathbf{t}, \mathrm{root}(\mathbf{t}))$.

Define $A = \{env, sub, result, lab\}$ and $D = \Phi_k \cup \{0, 1\}$. To every production $p = X \to r$ we associate the rule

$$\langle \sigma_0 = (env, sub, result, lab), \ldots, \sigma_{|r|} = (env, sub, result); R_p \rangle,$$

where the string language R_p is defined as follows: $\overline{\theta_0}\theta_0 o_0 \ell_0 \cdots \overline{\theta_n}\theta_n o_n \ell_n \in R_p$ if

1. $\overline{\theta_i}, \theta_i \in \Phi_k$, $o_i \in \{0, 1\}$, and $\ell_i \in N \cup T$ for $i = 1, \ldots, n$;
2. for $i = 1, \ldots, n$, if $\ell_i \in T$ then $\theta_i = \tau_k^{\mathrm{MSO}}(\mathbf{t}(\ell_i), \mathrm{root})$;
3. if $X = U$ then $\overline{\theta_0} = \tau_k^{\mathrm{MSO}}(\mathbf{t}(U), \mathrm{root})$;
4. there exists a tree $\mathbf{t}$ with a node $\mathbf{n}$ with children $\mathbf{n}_1, \ldots, \mathbf{n}_n$ such that $\tau_k^{\mathrm{MSO}}(\mathbf{t}_\mathbf{n}, \mathrm{root}) = \theta_0$, $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_\mathbf{n}}, \mathbf{n}) = \overline{\theta_0}$, and for $i = 1, \ldots, n$, $\tau_k^{\mathrm{MSO}}(\mathbf{t}_{\mathbf{n}_i}, \mathrm{root}) = \theta_i$ and $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_{\mathbf{n}_i}}, \mathbf{n}) = \overline{\theta_i}$;
5. for $i = 0, \ldots, n$, $o_i = 1$ if and only if there exists a tree $\mathbf{t}$ with a node $\mathbf{n}$ such that $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t}_\mathbf{n}}, \mathbf{n}) = \overline{\theta_i}$, $\tau_k^{\mathrm{MSO}}(\mathbf{t}_\mathbf{n}, \mathrm{root}) = \theta_i$, and $\mathbf{t} \models \varphi[\mathbf{n}]$;

We construct a two-way deterministic string automaton B with one pebble accepting R_p. By Proposition 2.9, R_p is regular. For steps (1–3,5), B just makes one pass through the input string. Recall, for instance, that, by Proposition 6.3(1), o_i only depends on $\overline{\theta_i}$ and θ_i. For step (4), B first simulates the automaton $M = (Q, \Phi_k, \delta, s_0, F)$ for $\mathrm{SUB}(\ell_0, \theta_0)$ of Lemma 6.5 on $\theta_1 \cdots \theta_n$. Hereafter it checks the consistency of $\overline{\theta_0}$, and $\overline{\theta_1}, \ldots, \overline{\theta_n}$. Note that for every $i = 1, \ldots, n$, by Proposition 6.3(3), $\overline{\theta_i}$ depends only on $\overline{\theta_0}$, θ_i, $\delta^*(s_0, \theta_1 \cdots \theta_{i-1})$ and $\delta^*(s_0, \theta_{i+1} \cdots \theta_n)$, where δ is the transition function of M. Hence, B remembers $\overline{\theta_0}$ in its state and then successively puts its pebble on each input tuple. If the pebble lays on the i-th tuple then M computes

$\delta^*(s_0, \theta_1 \cdots \theta_{i-1})$ and $\delta^*(s_0, \theta_{i+1} \cdots \theta_n)$, whereafter it returns to the pebble and checks whether $\xi_{\ell_0}(\overline{\theta_0}, \delta^*(s_0, \theta_1 \cdots \theta_{i-1}), \theta_i, \delta^*(s_0, \theta_{i+1} \cdots \theta_n)) = \overline{\theta_i}$ (ξ_{ℓ_0} is the function defined in the proof of Theorem 6.4).

It remains to show that for each tree only one valuation exists satisfying $\mathcal{F}$. Using Proposition 6.3(2), a simple induction on the height of nodes in the tree shows that $\tau_k^{\mathrm{MSO}}(\mathbf{t_n}, \mathbf{n}) = \theta$ whenever $v(sub(\mathbf{n})) = \theta$ for a valuation v satisfying $\mathcal{F}$. An induction on the depth of nodes in the tree, using the above and Proposition 6.3(3), then shows that $\tau_k^{\mathrm{MSO}}(\overline{\mathbf{t_n}}, \mathbf{n}) = \theta$ whenever $v(env(\mathbf{n})) = \theta$ for a valuation v satisfying $\mathcal{F}$.

This concludes the proof of the theorem. ■

10 Discussion

In other work [36], Schwentick and the present author defined query automata to query structured documents. Query automata are two-way automata over (un)ranked trees that can select nodes depending on the current state and on the label at these nodes. Query automata can express precisely the unary MSO definable queries and have an EXPTIME-complete equivalence problem. This makes them look rather similar to extended AGs. The two formalisms are, however, very different in nature. Indeed, query automata constitute a procedural formalism that has only local memory (in the state of the automaton), but which can visit each node more than a constant number of times. Attribute grammars, on the other hand, are a declarative formalism, whose evaluation visits each node of the input tree only a constant number of times (once for each attribute). In addition, they have a distributed memory (in the attributes at each node). It is precisely this distributed memory which makes extended AGs particularly well-suited for an efficient simulation of Region Algebra expressions. It is, therefore, not clear whether there exists an *efficient* translation from Region Algebra expressions into query automata.

Extended AGs can only express queries that retrieve subtrees from a document. It would be interesting to see whether the present formalism can be extended to also take restructuring of documents into account. A related paper in this respect is that of Crescenzi and Mecca [17]. They define an interesting formalism for the definition of wrappers that map derivation trees of regular grammars to relational databases. Their formalism, however, is only defined for regular grammars and the correspondence between actions

(i.e., semantic rules) and grammar symbols occurring in regular expressions is not so flexible as for extended AGs. Other work that uses attribute grammars in the context of databases includes work of Abiteboul, Cluet, and Milo [2] and Kilpeläinen et al. [27].

Acknowledgement

This paper is inspired by the work of Brüggemann-Klein, Murata and Wood. I thank them for providing me with a first draft of [11]. I thank Thomas Schwentick, Jan Van den Bussche, and Dirk Van Gucht for helpful discussions on topics treated in this paper.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[2] S. Abiteboul, S. Cluet, and T. Milo. A logical view of structured files. *VLDB Journal*, 7(2):96–114, 1998.

[3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[4] R. Beaza-Yates and G. Navarro. Integrating contents and structure in text retrieval. *ACM SIGMOD Record*, 25(1):67–79, March 1996.

[5] C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In P. Buneman C. Beeri, editor, *Database Theory – ICDT99*, volume 1540 of *Lecture Notes in Computer Science*, pages 296–313. Springer-Verlag, 1998.

[6] R. Bloem and J. Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. Technical Report 98-02, Rijksuniversiteit Leiden, January 1998. Revised version: `http://www.wi.leidenuniv.nl/home/engelfri`.

[7] R. Bloem and J. Engelfriet. Monadic second order logic and node relations on graphs and trees. volume 1261 of *Lecture Notes in Computer Science*, pages 144–161. Springer-Verlag, 1997.

[8] M. Blum and C. Hewitt. Automata on a 2-dimensional tape. In *Conference Record of 1967 Eighth Annual Symposium on Switching and Automata Theory*, pages 155–160. IEEE, 1967.

[9] R. Book, S. Even, S. Greibach, and G. Ott. Ambiguity in graphs and expressions. *IEEE Transactions on Computers*, c-20(2):149–153, 1971.

[10] A. Brüggemann-Klein and Wood D. One unambiguous regular languages. *Information and Computation*, 140(2):229–253, 1998.

[11] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree languages over non-ranked alphabets (draft 1). Unpublished manuscript, 1998.

[12] B. S. Chlebus. Domino-tiling games. *Journal of Computer and System Sciences*, 32(3):374–392, 1986.

[13] M. Consens and T. Milo. Optimizing queries on files. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, volume 23:2 of *SIGMOD Record*, pages 301–312. ACM Press, 1994.

[14] M. Consens and T. Milo. Algebras for querying text regions: Expressive power and optimization. *Journal of Computer and System Sciences*, 3:272–288, 1998.

[15] World Wide Web Consortium. Extensible Markup Language (XML). http://www.w3.org/XML/.

[16] B. Courcelle and P. Deransart. Proofs of partial correctness for attribute grammars with applications to recursive procedures and logic programming. *Information and Computation*, 78(1):1–55, 1988.

[17] V. Crescenzi and G. Mecca. Grammars have exceptions. *Information Systems – Special Issue on Semistructured Data*, 23(8):539–565, 1998.

[18] P. Deransart, M. Jourdan, and B. Lorho. *Attribute Grammars: Definition, Systems and Bibliography*, volume 323 of *Lecture Notes in Computer Science*. Springer, 1988.

[19] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1995.

[20] F. Gécseg and M. Steinby. Tree languages. In Rozenberg and Salomaa [41], chapter 1.

[21] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Informatica*, 25(4):355–423, 1988.

[22] N. Globerman and D. Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theoretical Computer Science*, 169(2):161–184, 1996.

[23] G.H. Gonnet and F.W. Tompa. Mind your grammar: a new approach to modelling text. In *Proceedings 13th Conference on VLDB*, pages 339–346, 1987.

[24] M. Gyssens, J. Paredaens, and D. Van Gucht. A grammar-based approach towards unifying hierarchical data models. *SIAM Journal on Computing*, 23(6):1093–1137, 1994.

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

[26] M. Jazayeri, W. F. Ogden, and W. C. Rounds. The intrinsically exponential complexity of the circularity problem for attribute grammars. *Communications of the ACM*, 18(12):697–706, 1975.

[27] P. Kilpeläinen, G. Lindén, H. Mannila, and E. Nikunen. A structured text database system. In R. Furuta, editor, *Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, The Cambridge Series on Electronic Publishing, pages 139–151. Cambridge University Press, 1990.

[28] P. Kilpeläinen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *Proceedings of the Sixteenth International Conference on Research and Development in Information Retrieval*, pages 214–222. ACM Press, 1993.

[29] P. Kilpeläinen and H. Mannila. Query primitives for tree-structured data. In M. Crochemore and D. Gusfield, editors, *Proceedings of the fifth Symposium on Combinatorial Pattern Matching*, pages 213–225. Springer-Verlag, 1994.

[30] D.E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968. See also *Mathematical Systems Theory*, 5(2):95–96, 1971.

[31] S. Maneth and F. Neven. A formalization of tree transformations in XSL. To appear in the proceedings of the Seventh International Workshop on Database Programming Languages, *Lecture Notes in Computer Science*, 1999.

[32] M. Murata. Forest-regular languages and tree-regular languages. Unpublished manuscript, 1995.

[33] A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In V. Arvind and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, pages 134–145. Springer, 1998.

[34] F. Neven. Structured document query languages based on attribute grammars: locality and non-determinism. In T. Ripke T. Polle and K.-D. Schewe, editors, *Fundamentals of Information Systems*, pages 129–142. Kluwer, 1998.

[35] F. Neven. *Design and Analysis of Query Languages for Structured Documents — A Formal and Logical Approach.* Doctor's thesis, Limburgs Universitair Centrum (LUC), 1999.

[36] F. Neven and T. Schwentick. Query automata. In *Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems*, pages 205–214. ACM Press, 1999.

[37] F. Neven and J. Van den Bussche. On implementing structured document query facilities on top of a DOOD. In F. Bry, R. Ramakrishnan, and K. Ramamohanarao, editors, *Deductive and Object-Oriented Databases*, volume 1341 of *Lecture Notes in Computer Science*, pages 351–367. Springer-Verlag, 1997.

[38] F. Neven and J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. In *Proceedings of the Seventeenth ACM Symposium on Principles of Database Systems*, pages 11–17. ACM Press, 1998.

[39] C. Pair and A. Quere. Définition et etude des bilangages réguliers. *Information and Control*, 13(6):565–593, 1968.

[40] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. Submitted, 1999.

[41] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 3. Springer, 1997.

[42] A. Salminen and F. Tompa. PAT expressions: an algebra for text search. *Acta Linguistica Hungarica*, 41:277–306, 1992.

[43] D. Suciu. Semistructured data and XML. In *Proceedings of the 5th International Conference on Foundations of Data Organization and Algorithms*, 1998.

[44] M. Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Information and Control*, 27(1):1–36, 1975.

[45] W. Thomas. Languages, automata, and logic. In Rozenberg and Salomaa [41], chapter 7.

[46] M. Y. Vardi. Automata theory for database theoreticians. In *Proceedings of the Eighth ACM Symposium on Principles of Database Systems*, pages 83–92. ACM Press, 1989.

[47] D. Wood. Standard generalized markup language: Mathematical and philosophical issues. In J. van Leeuwen, editor, *Computer Science Today. Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 344–365. Springer-Verlag, 1995.