Extensions of Attribute Grammars for Structured Document Queries

(extended abstract)

Frank Neven*

Limburgs Universitair Centrum, Universitaire Campus, Dept. WNI, Infolab,
B-3590 Diepenbeek, Belgium, E-mail: frank.neven@luc.ac.be

Abstract. Widely-used document specification languages like, e.g., SGML and 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, establish the complexity of their non-emptiness and equivalence problem to be complete for EXPTIME, and consider several extensions of extended AGs. As an application we show that the Region Algebra expressions introduced by Consens and Milo 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.

1 Introduction

Structured document databases can be seen as derivation trees of some grammar which functions as the "schema" of the database [2, 4, 22, 23, 25, 37, 42]. Document specification languages like, e.g., SGML [46] and XML [14], 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, 44], unranked trees only recently received new attention in the context of SGML and XML. Based on work of Pair and Quere [38] and Takahashi [43], Murata defined a bottom-up automaton model for unranked trees [30]. This required describing transition functions for an arbitrary number of children. Murata's approach is

* Research Assistant of the Fund for Scientific Research, Flanders.

DB $\rightarrow$ PoemList
PoemList $\rightarrow$ Poem PoemList
PoemList $\rightarrow$ Poem
Poem $\rightarrow$ VerseList
VerseList $\rightarrow$ Verse VerseList
VerseList $\rightarrow$ Verse
Verse $\rightarrow$ WordList
WordList $\rightarrow$ Word WordList
WordList $\rightarrow$ Word
Word $\rightarrow$ LetterList
LetterList $\rightarrow$ Letter LetterList
LetterList $\rightarrow$ Letter
Letter $\rightarrow$ a | ... | z

Fig. 1. A CFG modeling a list of poems.

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 [9].

The classical formalism of *attribute grammars*, introduced by Knuth [28], has always been a prominent framework for expressing computations on derivation trees. Therefore, in previous work, we investigated attribute grammars as a query language for derivation trees of CFGs [33, 36, 37]. 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). Attribute grammars are applied in such diverse fields of computer science as compiler construction and software engineering (for a survey, see [17]).

Inspired by the 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 $p = X \rightarrow r$ 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

[1] This is no loss of generality, as any regular language can be denoted by an unambiguous regular expression [7]. SGML is even more restrictive as it allows only *one*-unambiguous regular languages [8, 46].

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

Fig. 2. An ECFG modeling a list of poems.

AGs particularly relevant as a query language. Indeed, as argued by Suciu [42], achieving this capability is one of the major challenges when applying the techniques developed for semi-structured data [1, 10] 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 [3, 26, 27, 31, 32]. These are essentially unary queries: they map a document to a set of its nodes. Extended AGs are 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. The latter is also a lower bound since deciding non-circularity for standard attribute grammar is already known to be hard for EXPTIME [28, 24].
2. We generalize our earlier results on standard attribute grammars [5, 37] by showing that extended AGs express precisely the unary queries definable in monadic second-order logic (MSO).
3. 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.
4. We show that Region Algebra expressions (introduced by Consens and Milo [13]) can be simulated by extended AGs. Stated as such, the result is not 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 $2\hat{\ }(2\hat{\ }\ldots(2^n))$ 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 coNP lower bound [13].
5. We define *relational* extended AGs. These are a generalization of relational attribute grammars as introduced by Courcelle and Deransart [15]. 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

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

extended AGs 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.

Omitted proofs can be found in the author's Phd thesis [34].

2 Preliminaries

Let $\mathbf{N}$ denote the set of natural numbers. For a finite set S, we denote by $|S|$ the cardinality of S. For integers i and j, we denote by $[i,j]$ the set $\{i,\ldots,j\}$. In the following, Σ is a finite alphabet. If $w = a_1 \cdots a_n$ is a string over Σ then we denote a_i by $w(i)$ for $i \in \{1,\ldots,n\}$. We denote the length of w by $|w|$.

For a regular expression r over Σ, we denote by $L(r)$ the language defined by r and 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 $a(a + b^*)^* a$. Let $|r|$ denote the number of occurrences of Σ-symbols in r. For $i \in \{1,\ldots,|r|\}$, $r(i)$ denotes the Σ-symbol at the ith occurrence in r. Let $\widetilde{\Sigma}$ be the alphabet obtained from Σ by subscribing every symbol by all natural numbers, i.e., $\widetilde{\Sigma} := \{a_i \mid a \in \Sigma, i \in \mathbf{N}\}$. 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 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. [7], says that the restriction to unambiguous regular expressions is no loss of generality.

Proposition 2. *For every regular language R there exists an unambiguous regular expression r such that $L(r) = R$.*

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 ith letter in $\tilde{w}_r$. Intuitively, $\mathrm{pos}_r(i,w)$ indicates the position in r matching the ith 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,

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

In the sequel, when we say *regular expression*, we always mean *unambiguous* regular expression.

Extended AGs are defined over extended context-free grammars which are defined as follows. 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 $L(r_1) \cap L(r_2) \neq \emptyset$.

A *derivation tree* $\mathbf{t}$ over an ECFG G is a tree labelled with symbols from $N \cup T$ such that the root of $\mathbf{t}$ is labelled with U; for every interior node $\mathbf{n}$ with children $\mathbf{n}_1, \dots, \mathbf{n}_m$ there exists a production $X \to r$ such that $\mathbf{n}$ is labelled with X, for $i = 1, \dots, m$, $\mathbf{n}_i$ is labelled 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 labelled with a terminal. We denote by $\mathrm{root}(\mathbf{t})$ the root node of $\mathbf{t}$.

Note that derivation trees of ECFGs are *unranked* in the sense that the number of children of a node need not be bounded by any constant and does not depend on the label of that node.

Throughout the paper we make the harmless technical assumption that the start symbol does not occur on the right-hand side of a production.

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 [33, 36, 37]. 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 following rules then define the attribute *is_a* in the obvious way. Here, 0 refers to the left-hand side of the rule.

Consider the ECFG consisting of the sole rule $U \to (A+B)^*$. We now 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 labelled 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 now expresses the above query:

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

The 1 in $\mathit{select}(1)$ 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 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 is just the string

$$\begin{array}{ccccccccl} 1 & 1 & & 1 & 2 & 2 & 2 & & \text{position in } (A+B)^* \\ \mathbf{A} & \mathbf{A} & \# & \mathbf{A} & \mathbf{B} & \mathbf{B} & \mathbf{B} & & \\ 1 & 2 & & 3 & 4 & 5 & 6 & & \text{position in } AAABBB \end{array}$$

The attribute *select* of the third child will now be assigned the value *true* since the above string belongs to R_{true}. Note that $(B^*AB^*AB^*)^*$ 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

In this section we define extended attribute grammars (extended AGs) over ECFGs whose attributes can take only values from a finite set D. Hence, we leave the framework of only Booleans as attribute values. Nevertheless, we still have the equivalence with MSO as is shown in the next section.

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 3. An *attribute grammar vocabulary* is a tuple $(D, A, \mathrm{Syn}, \mathrm{Inh})$, 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 $A - \{lab\}$ such that for every $X \in N$, $\mathrm{Syn}(X) \cap \mathrm{Inh}(X) = \emptyset$; for every $X \in T$, $\mathrm{Syn}(X) = \emptyset$; and $\mathrm{Inh}(U) = \emptyset$.

If $a \in \mathrm{Syn}(X)$, we say that a is a *synthesized attribute of* X. If $a \in \mathrm{Inh}(X)$, 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 $i \in [1, |r|]$, define $p(i) = r(i)$. We fix some attribute grammar vocabulary $(A, D, \mathrm{Syn}, \mathrm{Inh})$ in the following definitions.

Definition 4. 1. Let $p = X \to r$ be a production of G and let a be an attribute of $p(i)$ for some $i \in [0, |r|]$. The triple (p, a, i) is called a *context* if $a \in \mathrm{Syn}(p(i))$ implies $i = 0$, and $a \in \mathrm{Inh}(p(i))$ 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 $j = [0, |r|]$, 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 5. 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 clear from the textual context which rule is associated to which context. We illustrate the above definitions with an example.

Example 6. 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. ∎

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

Fig. 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 7. 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 $\mathbf{n}$, $v((\mathit{lab}, \mathbf{n}))$ 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 8. 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 $\mathbf{n}_1, \ldots, \mathbf{n}_m$ be the children of $\mathbf{n}$ 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 $l \in [1, m]$, $j_l = \mathrm{pos}_r(l, w)$, 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 $l \in [1, m]$, $j_l = \mathrm{pos}_r(l, w)$, 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 9. 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 $\mathbf{n}$, $lab(\mathbf{n})$ 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, in the sequel, we always assume an extended AG to be non-circular.

Definition 10. The valuation $\mathcal{F}(\mathbf{t})$ equals $\mathcal{F}_l(\mathbf{t})$ with l such that $\mathcal{F}_l(\mathbf{t}) = \mathcal{F}_{l+1}(\mathbf{t})$.

We will use the following definition of a query:

Definition 11. A *query* is a function mapping each derivation tree to a set of its nodes.

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 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 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. ■

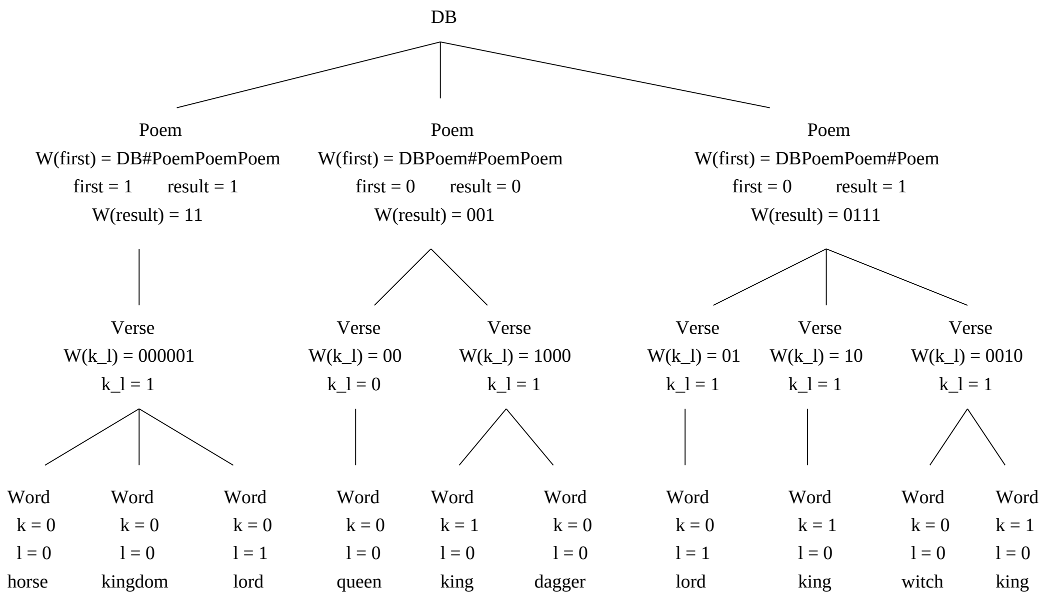

Fig. 4. A derivation tree and its valuation as defined by the extended AG in Figure 3.

5 Deciding non-circularity is in EXPTIME

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 [24], going from ranked to unranked 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 14. 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 exponentially increases the size of the AG. Indeed, a production $X \rightarrow (a+b)\cdots(a+b)$ (n times), for example, has to be translated to the set of productions $\{X \rightarrow w \mid w \in \{a,b\}^* \wedge |w| = N\}$. So, the complexity of the above algorithm is double exponential time. Therefore, we abandon this approach and give a different algorithm whose complexity is in EXPTIME.

To this end, we generalize the tree walking automata of Bloem and Engelfriet [6] to unranked trees. We then 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. Moreover, the size of $W_{\mathcal{F}}$ is polynomial in the size of $\mathcal{F}$. We then obtain our result by showing that testing whether a tree walking automaton cycles is in EXPTIME. We omit the details.

Theorem 15. *Deciding non-circularity of extended AGs is EXPTIME-complete.*

6 Expressiveness of extended AGs

In this section we characterize the expressiveness of extended AGs as the queries definable in monadic second-order logic.

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 $\mathbf{t}$. *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 (see, e.g., [19]). MSO can be used in the standard way to define queries.

Definition 16. Let $\varphi(x)$ be 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 obtain:

Theorem 17. *A query is expressible by an extended AG if and only if it is definable in MSO.*

We briefly sketch the proof idea. If $\mathcal{F}$ is an extended AG and $\mathbf{t}$ is a tree, then it can be shown that there exists only one assignment of values to the attributes of the nodes of $\mathbf{t}$ satisfying all semantic rules of $\mathcal{F}$. In MSO we then can use set variables to represent this assignment of values: for each $d \in D$ and $a \in A$, $Z_{a,d}$ will contain those nodes whose a-attribute is assigned the value d. We only have to verify that all semantic rules are satisfied under this assignment. This can easily be done since every regular language can be defined in MSO (see, e.g., the chapter by Thomas [44]). The result of the query expressed by $\mathcal{F}$ then consists of the nodes in $Z_{result,1}$. The other direction is more involved and is based on the ability of extended AGs to compute MSO-types of trees. See [34] for more details.

7 Optimization

We now obtain the exact complexity of some relevant optimization problems for extended AGs. These results will be used in the next section to obtain a new upper bound for deciding equivalence of Region Algebra expressions introduced by Consens and Milo [13].

We represent the regular languages R_d in the semantic rules by nondeterministic finite acceptors (NFAs). The size of an extended AG is the size of the attribute grammar vocabulary plus the size of the NFAs in the semantic rules. Consider the following optimization 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?

We obtain:

Theorem 18. NON-EMPTINESS *and* EQUIVALENCE *are EXPTIME-complete.*

We give a sketch of the proof.

1. NON-EMPTINESS: EXPTIME-hardness is shown by a reduction from TWO PLAYER CORRIDOR TILING (see Chlebus [11]).
   To show that NON-EMPTINESS is in EXPTIME, we transform a finite AG $\mathcal{F}$ into a non-deterministic bottom-up automaton $M_{\mathcal{F}}$ (NBTA) over unranked trees such that $M_{\mathcal{F}}$ accepts a tree if and only if $\mathcal{F}$ is non-empty. A detailed study of tree automata over unranked trees has been initiated by Brüggemann-Klein, Murata and Wood [9, 30]. The size of $M_{\mathcal{F}}$ is exponential in the size of $\mathcal{F}$ and the non-emptiness test of NBTAs can be done in PTIME. The latter is just a modification of the algorithm for testing non-emptiness for non-deterministic bottom-up tree automata over *ranked* trees (see, e.g., Vardi [45]). Hence, testing non-emptiness of extended AGs can be done in EXPTIME.
   The automaton $M_{\mathcal{F}}$ essentially guesses the values of the attributes (there are exponentially many since A and D are not fixed, hence $M_{\mathcal{F}}$ has exponentially many states) and verifies whether they satisfy all semantic rules. The difficulty in obtaining this, is to control the size of the NFAs. For example, to check consistency of the values of the inherited attributes we use two-way non-deterministic automata with one pebble (the pebble is used to remember where the symbol # should be). The sizes of these automata are polynomial in the size of $\mathcal{F}$. They are transformed into NFAs whose size is exponential in the size of $\mathcal{F}$ by using a construction of Globerman and Harel [21].
2. EQUIVALENCE: EXPTIME-hardness follows from EXPTIME-hardness of NON-EMPTINESS. We show that the complement of EQUIVALENCE is in EXPTIME. Given $\mathcal{F}_1$ and $\mathcal{F}_2$, we construct an extended AG $\mathcal{F}$ that operates in the following way: on input $\mathbf{t}$ it simulates both $\mathcal{F}_1$ and $\mathcal{F}_2$, it then selects every node that is selected by only one of $\mathcal{F}_1$ and $\mathcal{F}_2$. Hence, $\mathcal{F}_1$ is not equivalent to $\mathcal{F}_2$ if and only if $\mathcal{F}$ is non-empty. The extended AG $\mathcal{F}$ can clearly be constructed in LOGSPACE. As EXPTIME is closed under complement, EQUIVALENCE is in EXPTIME.

8 An application of extended AGs: optimization of Region Algebra expressions

The region algebra introduced by Consens and Milo [13, 12] is a set-at-a-time algebra, based on the PAT algebra [41], 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. Our algorithm therefore drastically improves the complexity of the equivalence test for the Region Algebra and matches more closely the coNP lower bound [13].

It should be pointed out that our definition differs slightly from the one in [13]. 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 [13] 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 19. 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 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 20. Consider the region index schema $\mathcal{I} = (\mathbf{Proc}, \mathbf{Func}, \mathbf{Var}, \Sigma)$. In Figure 5 an example of an instance over $\mathcal{I}$ is depicted. Here, $N_I = 16$, $I(\omega) =$ abcdefghijklmnop, $I(\mathbf{Proc}) = \{(1, 16), (6, 10)\}$, $I(\mathbf{Func}) = \{(12, 16)\}$ and $I(\mathbf{Var}) = \{(2, 3), (6, 7), (12, 13)\}$. ∎

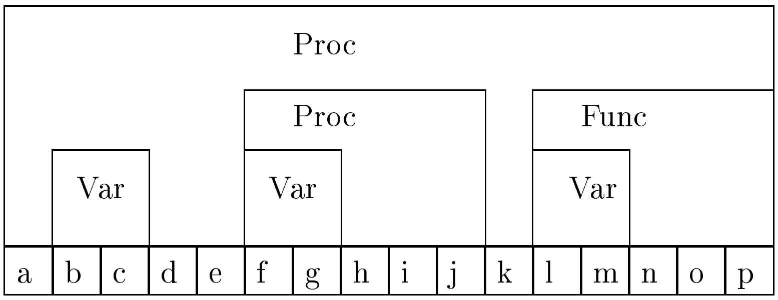

Fig. 5. An instance I over the region index schema of Example 20

For two regions r and s in I define:

– $r < s$ if $R(r) < L(s)$ (r *precedes* s); and
– $r \subset s$ 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 21. 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 5 is hierarchical. Like in [13], we only consider hierarchical instances. We now define the Region Algebra.

Definition 22. *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 23. 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. ■

The 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 now 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

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

For example, the derivation tree $\mathbf{t}_I$ of $G_{\mathcal{I}}$ representing the instance I of Figure 5 is depicted in Figure 6. 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.

Theorem 24. *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.*

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

Definition 25. 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 26. *Testing emptiness and equivalence of Region Algebra expressions is in* EXPTIME.

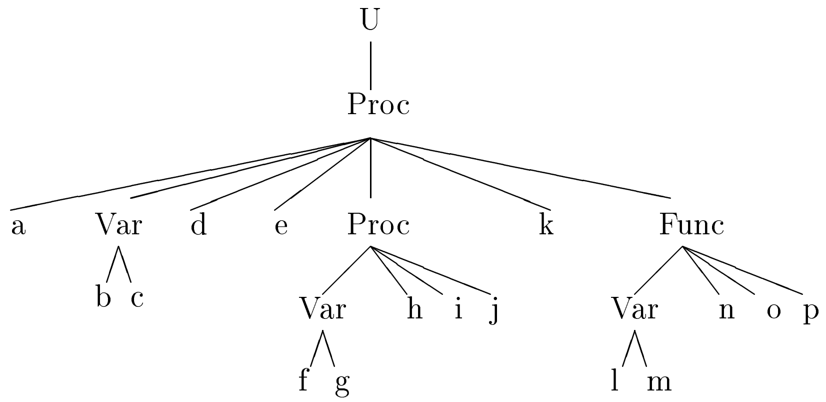

Fig. 6. The tree $\mathbf{t}_I$ corresponding to the instance I of Figure 5.

9 Relational extended AGs

Relational attribute grammars have been introduced by Courcelle and Deransart [15]. This concept is a generalization of standard attribute grammars, where the semantic rules do not specify functions, computing attributes in terms of other attributes, but rather specify relations among attributes. In this section we define relational extended AGs. To make a clear distinction between these and the extended AGs considered before, we will from now on refer to the latter as functional extended AGs. The main difference with 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. For one thing, this means that there is no longer a distinction between inherited and synthesized attributes. Further, attribute values are now defined implicitly. Hence, the result of an extended AG is no longer uniquely determined on every tree. We will show that relational extended AGs can express queries in various ways; and, although they are seemingly much less restrictive than functional extended AGs, we prove that they remain just as expressive.

We proceed with the definition of relational 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 $\mathbf{n}$, $v((\mathit{lab}, \mathbf{n}))$ 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 27. (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 28. Consider the ECFG of Example 6. 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 $\mathrm{DB} \to \mathrm{Poem}^+$, then define its associated rule as

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

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

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

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

We obtain that functional and relational extended AGs are equally expressive.

Theorem 29. *The class of queries expressed existentially (universally) by relational extended AGs coincides with the class of queries expressed by extended AGs.*

It suffices to 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 17. We omit further details.

10 Discussion

In other work [35], 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, hence, 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 [16]. 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. [25].

Finally, we compare extended AGs with the selective power of two query languages for XML. XML-QL is an SQL like query language for XML defined by Deutsch et al. [18] that allows to define general tree to tree transformations. The selective power of XML-QL, however, is restricted to regular *path* expressions. Consequently, it can only take the structure of paths into account, not of whole subtrees. E.g., queries like the one in Example 13: *retrieve every poem that has the words king or lord in every other verse* cannot be expressed. As another example we mention that XML-QL cannot select nodes whose children match a certain regular expressions. The latter functionality is obtained by Papakonstantinou and Vianu [39, 29] by introducing a form of horizontal navigation into their selection patterns. These patterns can easily be simulated by extended AGs, but still cannot express the query of example 13.

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 [9]. I thank Jan Van den Bussche and Dirk Van Gucht for helpful discussions on topics treated in this paper.

References

1. S. Abiteboul. Querying semi-structured data. In F. Afrati and Ph. Kolaitis, editors, *Database Theory—ICDT'97*, volume 1186 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 1997.
2. S. Abiteboul, S. Cluet, and T. Milo. A logical view of structured files. *VLDB Journal*, 7(2):96–114, 1998.
3. R. Beaza-Yates and G. Navarro. Integrating contents and structure in text retrieval. *ACM SIGMOD Record*, 25(1):67–79, March 1996.
4. 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.
5. R. Bloem and J. Engelfriet. Characterization of properties and relations defined in monadic second order logic on the nodes of trees. Technical Report 97-03, Rijksuniversiteit Leiden, 1997.
6. R. Bloem and J. Engelfriet. Monadic second order logic and node relations on graphs and trees. In J. Mycielski, G. Rozenberg, and A. Salomaa, editors, *Structures in Logic and Computer Science*, volume 1261 of *Lecture Notes in Computer Science*, pages 144–161. Springer-Verlag, 1997.
7. R. Book, S. Even, S. Greibach, and G. Ott. Ambiguity in graphs and expressions. *IEEE Transactions on Computers*, c-20(2):149–153, 1971.
8. A. Brüggemann-Klein and Wood D. One unambiguous regular languages. *Information and Computation*, 140(2):229–253, 1998.
9. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree languages over non-ranked alphabets (draft 1). Unpublished manuscript, 1998.
10. P. Buneman. Semistructured data. In *Proceedings of the Sixteenth ACM SIG-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 117–121. ACM Press, 1997.
11. B. S. Chlebus. Domino-tiling games. *Journal of Computer and System Sciences*, 32(3):374–392, 1986.
12. 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.
13. M. Consens and T. Milo. Algebras for querying text regions: Expressive power and optimization. *Journal of Computer and System Sciences*, 3:272–288, 1998.
14. World Wide Web Consortium. Extensible Markup Language (XML). http://www.w3.org/XML/.
15. 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.
16. V. Crescenzi and G. Mecca. Grammars have exceptions. *Information Systems – Special Issue on Semistructured Data*, 23(8):539–565, 1998.

17. P. Deransart, M. Jourdan, and B. Lorho. *Attribute Grammars: Definition, Systems and Bibliography*, volume 323 of *Lecture Notes in Computer Science*. Springer, 1988.
18. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: a query language for XML. In *Proceedings of the WWW8 Conference*, Toronto, 1999.
19. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1995.
20. F. Gécseg and M. Steinby. Tree languages. In Rozenberg and Salomaa [40], chapter 1.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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.
26. 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.
27. 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.
28. 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.
29. B. Ludäscher, Y.Papakonstantinou, P. Velikhov, and V. Vianu. View definition and DTD inference for XML. In *Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1999.
30. M. Murata. Forest-regular languages and tree-regular languages. Unpublished manuscript, 1995.
31. M. Murata. Data model for document transformation and assembly. In *Proceedings of the workshop on Principles of Digital Document Processing*, 1998. To appear in LNCS.
32. 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.
33. 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.
34. F. Neven. Phd thesis. In preparation, 1999.
35. 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.
36. 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.
37. 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.
38. C. Pair and A. Quere. Définition et etude des bilangages réguliers. *Information and Control*, 13(6):565–593, 1968.
39. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. Submitted, 1999.
40. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 3. Springer, 1997.
41. A. Salminen and F. Tompa. PAT expressions: an algebra for text search. *Acta Linguistica Hungarica*, 41:277–306, 1992.

42. D. Suciu. Semistructured data and XML. In *Proceedings of the 5th International Conference on Foundations of Data Organization and Algorithms*, 1998.
43. M. Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Information and Control*, 27(1):1–36, 1975.
44. W. Thomas. Languages, automata, and logic. In Rozenberg and Salomaa [40], chapter 7.
45. 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.
46. 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.