Expressiveness of Structured Document Query Languages Based on Attribute Grammars

(extended abstract)

Frank Neven*
Limburgs Universitair Centrum
fneven@luc.ac.be

Jan Van den Bussche
Limburgs Universitair Centrum
vdbuss@luc.ac.be

Abstract

Structured document databases can be naturally viewed as derivation trees of a context-free grammar. Under this view, the classical formalism of attribute grammars becomes a formalism for structured document query languages. From this perspective, we study the expressive power of BAGs: Boolean-valued attribute grammars with propositional logic formulas as semantic rules, and RAGs: relation-valued attribute grammars with first-order logic formulas as semantic rules. BAGs can express only unary queries; RAGs can express queries of any arity. We first show that the (unary) queries expressible by BAGs are precisely those definable in monadic second-order logic. We then show that the queries expressible by RAGs are precisely those definable by first-order inductions of linear depth, or, equivalently, those computable in linear time on a parallel machine with polynomially many processors. We finally show that RAGs are more powerful than monadic second-order logic for queries of any arity.

1 Introduction

As originally proposed by Gonnet and Tompa [GT87], a structured document database can be naturally viewed as a derivation tree over a context-free grammar. For instance, this is essentially the view taken by SGML [Gol95, Woo95].

The classical formalism of *attribute grammars*, introduced by Knuth [Knu68], 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). Attribute grammars are applied in such diverse fields of computer science as compiler construction and software engineering (for a survey, see [DJL88]).

Hence, it is natural to consider attribute grammars as a basis for structured document database languages. For instance, this approach was chosen by Abiteboul, Cluet and Milo [ACM93, ACM95]. Our goal in this paper is to understand the expressive power of attribute grammars as a structured document query language.

In the simple query facility provided by most information retrieval systems, a query amounts to the selection of certain nodes in the tree, corresponding to positions in the document or structural elements of the document, that are to be retrieved. In a previous paper [NVdB97], we proposed to use *Boolean-valued* attribute grammars (BAGs) to express such unary queries.[1] BAGs are attribute grammars with Boolean attribute values, and with propositional logic formulas as semantic rules. A BAG indeed expresses a query in a natural way: the result of the query expressed by a BAG consists of those nodes in the tree for which some designated attribute is true. Information retrieval systems usually query a set of structured documents instead of only one document. However, as far as query language design is concerned, a set of documents can be considered as one long structured document.

We will show that a unary query is expressible by a BAG if and only if it is definable in monadic second-order logic (MSO).[2] We found this pleasantly surprising, since at first it was not even clear to us that all *first*-order queries are BAG-expressible. The only-if direction is easy to prove. For the if direction, we make use of a classical theorem from the field of automata and logic (Doner-Thatcher-Wright [TW68, Don70]) stating that a tree language is recognizable by a finite tree automaton if and only if it is definable by an MSO-sentence. Using this theorem, we can prove our result by an intricate construction of a BAG which simulates, in parallel, the runs of a tree automaton on all possible Boolean labelings of a tree. An interesting corollary of our proof is that each BAG is equivalent to a BAG that can be evaluated in only two passes, the first one bottom-up and the second one top-down.

Having understood the expression of unary queries, we then turn to queries that result in relations among the nodes of the tree of arbitrary, fixed, arity. Thereto, we introduce *relation-valued* attribute grammars (RAGs), which use first-order logic formulas as semantic rules. The query expressed by a RAG is naturally defined as the value (a relation) of some designated attribute of the root. We show that the queries expressible by RAGs are precisely those definable by first-order inductions of linear depth. Results by Im-

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

Proceedings 17th ACM Symposium on Principles of Database Systems, ACM Press, 1998.

[1] We did not investigate expressiveness issues in that paper; this precisely is the purpose of the present paper.

[2] We must point out that this result was obtained independently by Bloem and Engelfriet [BE97].

merman [Imm89] imply that these are precisely the queries computable in linear time on a parallel random access machine with polynomially many processors.

Finally, we show that RAGs are strictly more powerful than monadic second-order logic, for queries of any arity. This implies in particular that even when restricting attention to unary queries, RAGs are more powerful than BAGs.

This paper is further organized as follows. After recalling the definition of a BAG, we investigate the expressive power of BAGs in Section 3. RAGs are introduced in Section 4, and their expressive power is characterized in Section 5. In Section 6, we discuss the relationship between RAGs and MSO. Concluding remarks are presented in Section 7.

2 Boolean-valued attribute grammars

For all what follows in this paper, we fix a context-free grammar $G = (N, T, P, U)$, where N is the set of non-terminals, T is the set of terminals, P is the set of productions, and U is the start symbol. We make the harmless technical assumption that the start symbol U does not appear on the right-hand side of any production.

Definition 2.1 An *attribute grammar vocabulary* has the form $(A, \mathrm{Syn}, \mathrm{Inh}, \mathrm{Att})$, where

- A is a finite sets of symbols called *attributes*;
- Syn, Inh, and Att are functions from $N \cup T$ to the powerset of A 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 $\mathrm{Inh}(U) = \emptyset$.
- for every X, $\mathrm{Att}(X) = \mathrm{Syn}(X) \cup \mathrm{Inh}(X)$.

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. The above conditions express that an attribute cannot be synthesized and inherited at the same time, that terminal symbols do not have synthesized attributes, and that the start symbol does not have inherited attributes.

We fix some attribute grammar vocabulary in the following definitions.

Definition 2.2 1. Let $p : X_0 \to X_1 \dots X_n$ be a production in P, and a an attribute of X_i for some $i \in \{0, \dots, n\}$. Then the triple (p, a, i) is called a *context* if $a \in \mathrm{Syn}(X_i)$ implies $i = 0$ and $a \in \mathrm{Inh}(X_i)$ implies $i > 0$.

2. A *rule in the context* (p, a, i) is an expression of the form
$$a(i) := \varphi,$$
where φ is a propositional logic formula over the set of proposition symbols
$$\{b(j) \mid j \in \{0, \dots, n\} \text{ and } b \in \mathrm{Att}(X_j)\}.$$

A BAG is then defined as follows:

Definition 2.3 A *Boolean-valued attribute grammar (BAG)* consists of an attribute grammar vocabulary, together with a mapping assigning to each context a rule in that context.

Example 2.4 In Figure 1 a simple example of a grammar and a BAG over this grammar are depicted. We have $\mathrm{Syn}(S) = \{\mathit{result}, \mathit{even}\}$, $\mathrm{Inh}(S) = \{\mathit{x\_before}\}$, $\mathrm{Syn}(B) = \{\mathit{is\_x}\}$, and $\mathrm{Att}(U) = \mathrm{Att}(x) = \mathrm{Att}(y) = \mathrm{Inh}(B) = \emptyset$. The semantics of this BAG will be explained below. ■

$U \to S$	$\mathit{x\_before}(1) := \text{false}$
$S \to BS$	$\mathit{x\_before}(2) := \mathit{is\_x}(1) \vee \mathit{x\_before}(0)$
	$\mathit{even}(0) := \neg \mathit{even}(2)$
	$\mathit{result}(0) := \mathit{even}(0) \wedge \mathit{x\_before}(0)$
$S \to B$	$\mathit{even}(0) := \text{false}$
	$\mathit{result}(0) := \text{false}$
$B \to x$	$\mathit{is\_x}(0) := \text{true}$
$B \to y$	$\mathit{is\_x}(0) := \text{false}$

Figure 1: Example of a BAG.

The semantics of a BAG is that it defines Boolean attributes of the nodes of derivation trees of the underlying grammar G. This is formalized next.

Definition 2.5 Let $\mathbf{t}$ be a derivation tree of G. A *valuation of* $\mathbf{t}$ is a function that maps each pair $(\mathbf{n}, a)$, where $\mathbf{n}$ is a node in $\mathbf{t}$ labeled X and a is an attribute of X, to a truth value (0 or 1).

In the sequel, for a pair $(\mathbf{n}, a)$ as above we will use the more intuitive notation $a(\mathbf{n})$.

Given a BAG $\mathcal{B}$ and a derivation tree $\mathbf{t}$, we construct a valuation $\mathcal{B}(\mathbf{t})$ of $\mathbf{t}$ by the following natural process. Initially, $\mathcal{B}(\mathbf{t})$ is nowhere defined. We then repeatedly extend $\mathcal{B}(\mathbf{t})$ as follows:

> Choose a node $\mathbf{n}_0$ in $\mathbf{t}$ labeled X_0, with children $\mathbf{n}_1$, ..., $\mathbf{n}_n$ labeled X_1, ..., X_n respectively. Let $p : X_0 \to X_1 \dots X_n$ be the corresponding production of G. Choose a rule $a(i) := \varphi$ in some context (p, a, i), such that for each proposition symbol $b(j)$ occurring in φ, the truth value of $b(\mathbf{n}_j)$ in $\mathcal{B}(\mathbf{t})$ has already been determined. Then we can evaluate φ under this truth assignment and define the truth value of $a(\mathbf{n}_i)$ in $\mathcal{B}(\mathbf{t})$ as the result of this evaluation.

If this process results in a totally defined valuation $\mathcal{B}(\mathbf{t})$ for every tree $\mathbf{t}$, we call $\mathcal{B}$ *non-circular*.[3] From now on, we will only consider BAGs that are non-circular. (Non-circularity is well known to be decidable [Knu68].)

It is well known that the evaluation of an attribute grammar takes linear time when counting the evaluation of a semantic rule as one unit of time (see, e.g., [DJL88]). Since a fixed propositional formula can indeed be evaluated in constant time, the valuation $\mathcal{B}(\mathbf{t})$ of a BAG $\mathcal{B}$ on a tree $\mathbf{t}$ can thus be computed in time linear in the size of $\mathbf{t}$.

Definition 2.6 A *unary query* is a function, mapping each derivation tree to a set of its nodes.

In this section and in Section 3, query means unary query. A BAG $\mathcal{B}$ can be used in a simple way to express queries. Among the attributes in the vocabulary of $\mathcal{B}$, we designate some attribute *result*, and define:

Definition 2.7 A BAG $\mathcal{B}$ expresses the query $\mathcal{Q}$ defined by
$$\mathcal{Q}(\mathbf{t}) = \{\mathbf{n} \mid \mathit{result}(\mathbf{n}) \text{ is true in the valuation } \mathcal{B}(\mathbf{t})\},$$
for every tree $\mathbf{t}$.

[3] This definition is equivalent to the more usual definition of non-circularity in terms of dependency graphs [Knu68].

Example 2.8 Recall the BAG of Figure 1. A derivation tree of the underlying grammar can be viewed naturally as a string over the alphabet $\{x, y\}$. Every node labeled S in the tree represents a position in the string. Now consider the semantic rules defining the synthesized attribute *even*. They can be evaluated bottom-up; for any node $\mathbf{n}$, *even*$(\mathbf{n})$ is true iff $\mathbf{n}$ is even-numbered when counting up from the bottom. The semantic rules defining the inherited attribute *x_before* can be evaluated top-down; *x_before*$(\mathbf{n})$ is true iff the letter x occurs in the string somewhere before position $\mathbf{n}$. Finally, the semantic rules for the attribute *result* simply define *result*$(\mathbf{n})$ as *x_before*$(\mathbf{n}) \wedge$ *even*$(\mathbf{n})$. Hence, the BAG expresses the query retrieving those even-numbered positions that come after an x in the string. An illustration is given in Figure 2. ■

More realistic examples of structured document queries expressed using BAGs can be found in our previous paper [NVdB97].

3 Expressive power of BAGs

In this section we want to characterize the expressive power of BAGs as a query language in terms of logic. Thereto, we naturally view derivation trees of the grammar G as finite structures (in the sense of logic [End72]) over the vocabulary consisting of all non-terminal and terminal symbols of G as unary relation symbols, and the binary relation symbols $S_1, \ldots, S_r$, where r is the maximal length of the right-hand side of a production of G.

The domain of a tree $\mathbf{t}$, viewed as a structure, equals the set of nodes of $\mathbf{t}$. For each $i = 1, \ldots, r$, the relation S_i in $\mathbf{t}$ equals the set of pairs $(\mathbf{n}, \mathbf{n}')$ such that $\mathbf{n}'$ is the i-th child of $\mathbf{n}$ in $\mathbf{t}$. Finally, for each terminal or non-terminal X, the relation X in $\mathbf{t}$ equals the set of X-labeled nodes in $\mathbf{t}$.

The logics we will be using are standard first-order logic, and its extensions *monadic second-order logic (MSO)* and *partial fixpoint logic*. Monadic second-order logic allows the use of *set variables* ranging over the sets of nodes of a tree, in addition to the individual variables ranging over the nodes themselves as provided by first-order logic. Partial fixpoint logic allows first-order logic formulas to be iterated. An exposition on these logics can be found in, e.g., Ebbinghaus and Flum's book [EF95].

In the present section, we focus on MSO. An MSO-formula $\varphi(x)$, with x an individual variable, defines a unary query $\mathcal{Q}$ in the standard way: for any tree $\mathbf{t}$,

$$\mathcal{Q}(\mathbf{t}) := \{\mathbf{n} \in \mathbf{t} \mid \mathbf{t} \models \varphi[\mathbf{n}]\}.$$

We establish:

Theorem 3.1 *A query is expressible by a BAG if and only if it is definable in monadic second-order logic.*

Proof. (Sketch) *Only if.* The BAG evaluation process can be naturally simulated as the fixpoint of a system of first-order logic formulas. The induction variables of this system are the set variables X_a for each attribute a; X_a stands for the currently computed set of nodes for which attribute a is true. For each a there is a formula φ_a defining the new value of X_a from the old values of the X_bs. The formula φ_a is not necessarily positive. However, by an induction based on the non-circularity of the BAG, one can show that iterating this system of formulas leads to a least fixpoint which coincides with the semantics of the BAG.[4] Now clearly the least fixpoint of a system of first-order formulas with monadic induction variables is definable in MSO, as had to be proven.

If. The proof uses tree automata [GS84, Var89]. A (bottom-up deterministic) tree automaton $\mathcal{M} = (Q, \delta, F)$, working on derivation trees of context-free grammar G, consists of a finite set Q of states, a set $F \subseteq Q$ of final states, and a transition function δ mapping the terminal symbols of G, as well as all tuples of the form $(p, q_1, \ldots, q_n)$, where $p = X_0 \to X_1 \ldots X_n$ is a production of G and $q_1, \ldots, q_n$ are states, to states. The function δ gives rise to a function $\overline{\delta}$ mapping derivation trees and their subtrees to states. A tree $\mathbf{t}$ consisting of one single node labeled by a terminal symbol ℓ is mapped to $\delta(\ell)$. A tree of the form

$$\begin{array}{ccc} & X_0 & \\ \swarrow & \cdots & \searrow \\ X_1 & & X_n \\ \mathbf{t}_1 & & \mathbf{t}_n \end{array}$$

is mapped to

$$\delta(X_0 \to X_1 \ldots X_n, \overline{\delta}(\mathbf{t}_1), \ldots, \overline{\delta}(\mathbf{t}_n)).$$

A derivation tree $\mathbf{t}$ is said to be *accepted* by the automaton $\mathcal{M}$ if $\overline{\delta}(\mathbf{t}) \in F$. The set of all trees accepted by $\mathcal{M}$ is denoted by $L(\mathcal{M})$ (called the tree language defined by $\mathcal{M}$).

Consider a tree $\mathbf{t}$ and a set $\mathbf{s}$ of nodes of $\mathbf{t}$. We can view the pair $(\mathbf{t}, \mathbf{s})$ as a labeling of $\mathbf{t}$ with zeros and ones by labeling the nodes in $\mathbf{s}$ with 1 and the nodes not in $\mathbf{s}$ with 0. So if $\mathbf{t}$ is a derivation tree of grammar G, then $(\mathbf{t}, \mathbf{s})$ is a derivation tree of the annotated grammar G^{01} having as symbols all those of the form $X0$ and $X1$, with X a symbol of G, and having as productions all those of the form $X_0 i_0 \to X_1 i_1 \ldots X_n i_n$, with $X_0 \to X_1 \ldots X_n$ a production of G and $i_0, \ldots, i_n \in \{0, 1\}$.

To prove the theorem, consider an MSO formula $\varphi(z)$ with one free individual variable z. Define the MSO formula $\varphi'(Z)$, having one free set variable Z but no free individual variables, as $\varphi' := (\forall z)(Z(z) \leftrightarrow \varphi(z))$. By the above, the set $\mathcal{L}_{\varphi'} := \{(\mathbf{t}, \mathbf{s}) \mid \mathbf{t} \models \varphi'[\mathbf{s}]\}$ can be viewed as a set of 0-1-labeled trees. Now by a classical theorem (Doner-Thatcher-Wright, [Don70, TW68]), for every MSO formula $\psi(Z)$ there exists a tree automaton $\mathcal{M}_\psi$ (defined over G^{01}) such that $L(\mathcal{M}_\psi) = \mathcal{L}_\psi$.[5] In particular, this holds for φ'.

So there exists an automaton $\mathcal{M} = \mathcal{M}_{\varphi'}$ accepting precisely those pairs $(\mathbf{t}, \mathbf{s})$ for which $\mathbf{t} \models \varphi'[\mathbf{s}]$. Note that we have constructed φ' in such a way that for every tree $\mathbf{t}$ there is exactly one set $\mathbf{s}$ for which $\mathbf{t} \models \varphi'[\mathbf{s}]$, namely $\mathbf{s} = \{\mathbf{n} \mid \mathbf{t} \models \varphi[\mathbf{n}]\}$. In other words, for each tree $\mathbf{t}$, the automaton $\mathcal{M}$ will accept exactly one 0-1-labeling of $\mathbf{t}$, and this "accepted labeling" labels with 1 precisely those nodes in the result on $\mathbf{t}$ of the query defined by φ.

Hence, the theorem is proved if we can construct a BAG $\mathcal{B}$ with an attribute *result*, such that for each tree $\mathbf{t}$ and each node $\mathbf{n}$ of $\mathbf{t}$, *result*$(\mathbf{n})$ is true in $\mathcal{B}(\mathbf{t})$ iff $\mathbf{n}$ is labeled 1 in the labeling of $\mathbf{t}$ accepted by $\mathcal{M}$. This can be achieved by simulating the execution of $\mathcal{M}$ on all possible 0-1-labelings of $\mathbf{t}$. (It is important to realize that $\mathcal{B}$ must be defined over the original grammar G, while $\mathcal{M}$ is defined over the annotated grammar G^{01}.)

The BAG behaves as follows. In a first, bottom-up, pass over the tree, synthesized attributes *can-q* are defined such that for each node $\mathbf{n}$, *can-q*$(\mathbf{n})$ is true in $\mathcal{B}(\mathbf{t})$ iff $\mathcal{M}$ assumes

[4] We have exploited this idea further in our previous paper [NVdB97], where we investigated BAG evaluation mechanisms using deductive rules.

[5] Actually, this does not really follow from the Doner-Thatcher-Wright theorem itself, but rather from the standard proof of this theorem [Tho97].

$$
\begin{array}{ccccccccccc}
 & & \begin{array}{c} even = 0 \\ x\_before = 0 \\ result = 0 \end{array} & & \begin{array}{c} even = 1 \\ x\_before = 0 \\ result = 0 \end{array} & & \begin{array}{c} even = 0 \\ x\_before = 0 \\ result = 0 \end{array} & & \begin{array}{c} even = 1 \\ x\_before = 1 \\ result = 1 \end{array} & & \begin{array}{c} even = 0 \\ x\_before = 1 \\ result = 0 \end{array} \\
U & \longrightarrow & S & \longrightarrow & S & \longrightarrow & S & \longrightarrow & S & \longrightarrow & S \\
 & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
 & & B \;\; is\_x = 0 & & B \;\; is\_x = 0 & & B \;\; is\_x = 1 & & B \;\; is\_x = 0 & & B \;\; is\_x = 1 \\
 & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
 & & y & & y & & x & & y & & x
\end{array}
$$

Figure 2: A derivation tree and its valuation defined by the BAG of Figure 1.

state q at $\mathbf{n}$ in its execution on some labeling of $\mathbf{t}$. Since the accepted labeling is unique, there is only one final state q_F such that $can\text{-}q_F(\mathbf{r})$ is true, where $\mathbf{r}$ is the root of $\mathbf{t}$. So we can define synthesized attributes *must-q* of U (the start symbol) such that $must\text{-}q_F(\mathbf{r})$ is true and, for all other states q, $must\text{-}q(\mathbf{r})$ is false. Now in a second, top-down, pass over the tree, inherited attributes *must-q* are defined on all other nodes, such that for each node $\mathbf{n}$, $must\text{-}q(\mathbf{n})$ is true iff in a possible execution $\mathcal{M}$ assumes q at $\mathbf{n}$ and q' at the parent $\mathbf{p}$ of $\mathbf{n}$, where q' is the state such that $must\text{-}q'(\mathbf{p})$ is true. One can show that $\mathcal{M}$ assumes state q at $\mathbf{n}$ on the accepted labeling if and only if $must\text{-}q(\mathbf{n})$ is true. Hence, after all attributes *must-q* are defined for a particular node, we can set the desired value for its attribute *result*. We omit the formal details. ∎

As a corollary of our proof of Theorem 3.1 we obtain a normal form for BAGs:

Corollary 3.2 *Every BAG is equivalent to one which can be evaluated in two passes.*[6]

4 Relation-valued attribute grammars

In this section, we generalize BAGs to *relation*-valued attribute grammars (RAGs).

Example 4.1 As a first example, consider the RAG shown in Figure 3. A derivation tree of the underlying grammar models a set (S) of documents (D). Each document is a list (L) of paragraphs (p). The synthesized attribute *result* of U and L is relation-valued; on any tree, the value of *result* at the root will be the ternary relation consisting of all triples (d, f, l) such that d is a document, f is the first paragraph of d, and l is the last paragraph of d. This relation is computed using the synthesized attributes *first* and *last* of D and L; for every document node $\mathbf{n}$, $\mathit{first}(\mathbf{n})$ contains the first paragraph of that document, and $\mathit{last}(\mathbf{n})$ contains the last. These attributes are computed in turn using the inherited attribute *begin* and the synthesized attribute *end* of L, which are Boolean-valued; for any L-node $\mathbf{n}$, $\mathit{begin}(\mathbf{n})$ is true if $\mathbf{n}$ marks the beginning of a document, and $\mathit{end}(\mathbf{n})$ is true if $\mathbf{n}$ marks the end. Note that we now use first-order expressions, rather than propositional ones, to define the values of the attributes. ∎

Let us indicate the differences between BAGs and RAGs more formally.

To each attribute a we now associate an arity r_a (a natural number). A rule in the context (p, a, i), with $p : X_0 \to X_1 \dots X_n$ now becomes an expression of the form

$$a(i) := \varphi(x_1, \dots, x_{r_a}),$$

$U \to S$	$\mathit{result}(0) := \mathit{result}(1)$
$S \to DS$	$\mathit{result}(0) := (\{(1)\} \times \mathit{first}(1) \times \mathit{last}(1)) \cup \mathit{result}(2)$
$S \to$	$\mathit{result}(0) := \emptyset$
$D \to L$	$\mathit{first}(0) := \mathit{first}(1)$ $\mathit{last}(0) := \mathit{last}(1)$ $\mathit{begin}(1) :=$ true
$L \to pL$	$\mathit{first}(0) :=$ **if** $\mathit{begin}(0)$ **then** $\{(1)\}$ **else** $\emptyset$ $\mathit{last}(0) :=$ **if** $\mathit{end}(2)$ **then** $\{(1)\}$ **else** $\mathit{last}(2)$ $\mathit{begin}(2) :=$ false $\mathit{end}(0) :=$ false
$L \to$	$\mathit{end}(0) :=$ true $\mathit{first}(0) := \emptyset$ $\mathit{last}(0) := \emptyset$

Figure 3: Example of a RAG.

where φ is a first-order logic formula over the vocabulary

$$\bigcup_{0 \le j \le n} \{b(j) \mid b \in \mathrm{Att}(X_j)\} \cup \{0, 1, \dots, n\},$$

where for each $j = 0, \dots, n$, $b(j)$ is a relation symbol of arity r_b, and j is a constant symbol. A valuation of a derivation tree $\mathbf{t}$ is now a function that maps each pair $(\mathbf{n}, a)$, where $\mathbf{n}$ is a node labeled X and a is an attribute of X, to an r_a-ary relation over the nodes of $\mathbf{t}$. Given a RAG $\mathcal{R}$ and a derivation tree $\mathbf{t}$, we construct a valuation $\mathcal{R}(\mathbf{t})$ of $\mathbf{t}$ by the following natural process. Initially, $\mathcal{R}(\mathbf{t})$ is nowhere defined. We then repeatedly extend $\mathcal{R}(\mathbf{t})$ as follows:

> Choose a node $\mathbf{n}_0$ in $\mathbf{t}$ labeled X_0, with children $\mathbf{n}_1, \dots, \mathbf{n}_n$ labeled $X_1, \dots, X_n$ respectively. Let $p : X_0 \to X_1 \dots X_n$ be the corresponding production of G. Choose a rule $a(i) := \varphi$ in some context (p, a, i), such that for each relation symbol $b(j)$ occurring in φ, the relation $b(\mathbf{n}_j)$ has already been determined by $\mathcal{R}(\mathbf{t})$. We compute the relation defined by φ, where we interpret each relation symbol $b(j)$ by the relation $b(\mathbf{n}_j)$ and each constant symbol j by the node $\mathbf{n}_j$. All variables used in φ range over the set of nodes of $\mathbf{t}$. The relation $a(\mathbf{n}_i)$ in $\mathcal{R}(\mathbf{t})$ is now defined as the result of this computation.

A *(k-ary) query* is a function which maps a tree to a (k-ary) relation on the nodes of that tree. A RAG can be used to express queries in a simple way. Among the attributes of the start symbol U we designate some attribute *result*, and define:

Definition 4.2 A RAG $\mathcal{R}$ expresses the query $\mathcal{Q}$ defined as follows: for any tree $\mathbf{t}$, $\mathcal{Q}(\mathbf{t})$ equals the value of $\mathit{result}(\mathbf{r})$ in $\mathcal{R}(\mathbf{t})$, where $\mathbf{r}$ is the root of $\mathbf{t}$.

[6] In the literature, attribute grammars having the latter property are known as *simple-2-pass* attribute grammars [DJL88].

Each BAG-expressible unary query is also RAG-expressible. Indeed, Boolean values and propositional logic are trivial cases of relations and first-order logic. So every BAG is also a RAG. Recall that to express a unary query, a BAG marks all nodes in the result with the value true. In contrast, from a RAG we expect the result to be present as a relation at the root. Clearly a RAG simulating a BAG can bring the nodes marked by the BAG up to the root using a synthesized set-valued attribute.

5 Expressive power of RAGs

In this section, we relate RAGs to the logic PFP[n] of linearly-bounded iterations of first-order formulas. To define this logic more formally, let $\varphi(x_1, \ldots, x_k, X)$ be a first-order logic formula over the vocabulary corresponding to the context-free grammar under consideration (as explained in the beginning of Section 3). The x_is are free individual variables; X is a k-ary relation variable that can be used in φ in addition to the relation symbols provided by the vocabulary.

On any tree $\mathbf{t}$, consider the stages $\varphi^n(\mathbf{t})$ obtained by iterating φ starting with the empty relation for X. So

$$\varphi^1(\mathbf{t}) := \{(\mathbf{n}_1, \ldots, \mathbf{n}_k) \mid \mathbf{t} \models \varphi[\mathbf{n}_1, \ldots, \mathbf{n}_k, \emptyset]\},$$

and

$$\varphi^{i+1}(\mathbf{t}) := \{(\mathbf{n}_1, \ldots, \mathbf{n}_k) \mid \mathbf{t} \models \varphi[\mathbf{n}_1, \ldots, \mathbf{n}_k, \varphi^i]\}.$$

If φ is such that on every tree these stages converge to a fixpoint after a number of iterations that is linear in the number of nodes of the tree, then we say that φ is *linearly bounded* and denote the fixpoint by the expression PFP[φ].

We can now define the logic PFP[n] simply as follows: formulas are constructed just as in first-order logic, with the addition that we also allow atomic formulas of the form PFP[φ]$(x_1, \ldots, x_k)$ with φ linearly bounded. The semantics is as explained above.

A PFP[n]-formula $\psi(y_1, \ldots, y_\ell)$ defines an ℓ-ary query $\mathcal{Q}$ in the standard way: for any tree $\mathbf{t}$,

$$\mathcal{Q}(\mathbf{t}) := \{(\mathbf{n}_1, \ldots, \mathbf{n}_\ell) \mid \mathbf{t} \models \psi[\mathbf{n}_1, \ldots, \mathbf{n}_\ell]\}.$$

We establish:

Theorem 5.1 *A query is expressible by a RAG if and only if it is definable in* PFP[n].

Proof. (Sketch) *Only if.* As was the case for BAGs (cf. proof of Theorem 3.1), the RAG evaluation process can again be naturally simulated as the fixpoint of a system of first-order logic formulas. The induction variables of this system are the relation variables X_a for each attribute a; X_a now stands for the set of tuples $(\mathbf{n}, \mathbf{n}_1, \ldots, \mathbf{n}_r)$, where $\mathbf{n}$ is a node and $(\mathbf{n}_1, \ldots, \mathbf{n}_r)$ is a tuple in the currently computed value for $a(\mathbf{n})$. For each a there is a formula φ_a defining the new value of X_a from the old values of the X_bs. By an induction based on the non-circularity of the BAG, one can show that iterating this system of formulas leads to a fixpoint which is reached after a linear number of iterations and which coincides with the semantics of the RAG. By the Simultaneous Induction Lemma [Mos74, EF95], we can simulate the fixpoint computation of a system of formulas by the fixpoint computation of a single formula without changing the number of needed iterations. Hence, each query expressed by a RAG can be defined in PFP[n].

If. The crux of the proof is the simple observation that RAGs can compute all the relations that make up a derivation tree, viewed as a relational structure, in one bottom-up pass over the tree. In a subsequent top-down pass, we can make these relations available at all nodes. A linearly-bounded iteration of a first-order formula can now be simulated in one traversal of the tree, where the different stages are passed over as relational attribute values. We omit the formal details (which are rather complicated). ∎

The relevance of PFP[n] was indicated by Immerman [Imm89], who showed that for any polynomial time bound $t(n)$, PFP[$t(n)$] captures the complexity class CRAM[$t(n)$] consisting of all problems decidable in time $O(t(n))$ by a parallel machine with polynomially many processors. Hence, we have:

Corollary 5.2 *A query is expressible by a RAG if and only if it is computable in linear parallel time with polynomially many processors.*

To be precise, we should note that Immerman did not really consider PFP[$t(n)$], but rather LFP[$t(n)$], i.e., he did not consider partial fixpoints (fixpoints of arbitrary formulas) but least fixpoints (fixpoints of positive formulas).[7] This does not make a difference in our context, however, since PFP[n] and LFP[n] are equivalent on ordered structures. This brings us to a second point: when capturing complexity classes using logic, one typically assumes structures to be ordered. However, also this does not pose a problem in our context, since RAGs can compute an order on derivation trees, as illustrated by the following example.

Example 5.3 Consider the RAG shown in Figure 4, over a context-free grammar for binary trees. The query expressed by the RAG results on each tree in a linear order on its nodes, corresponding to a postorder traversal [Knu82] of the tree. This example can easily be generalized to arbitrary grammars. ∎

6 RAGs versus MSO

We have now characterized BAGs as the unary queries definable in MSO (Theorem 3.1), and RAGs as the queries (of arbitrary arity) definable in PFP[n] (Theorem 5.1). It remains to compare these two formalisms with each other. We will now show that RAGs are actually strictly more powerful than MSO.

Theorem 6.1 *Every query definable in MSO is expressible by a RAG using only synthesized attributes.*

Proof. (Sketch) Consider an MSO formula $\varphi(x_1, \ldots, x_k)$. For any tree $\mathbf{t}$ and nodes $\mathbf{n}_1, \ldots, \mathbf{n}_k$ of $\mathbf{t}$, we can view the tuple $(\mathbf{t}, \mathbf{n}_1, \ldots, \mathbf{n}_k)$ as a labeling of $\mathbf{t}$ with elements of $\{0,1\}^k$ by labeling a node $\mathbf{n}$ with $u_1 \ldots u_k$ such that for $i = 1, \ldots, k$, $u_i = 1$ if $\mathbf{n} = \mathbf{n}_i$ and $u_i = 0$ otherwise. So $(\mathbf{t}, \mathbf{n}_1, \ldots, \mathbf{n}_k)$ is a derivation tree of an annotated grammar G^k which we will not define formally.

It is easy to write an MSO sentence ψ over G^k defining the set $\mathcal{L}_\varphi := \{(\mathbf{t}, \mathbf{n}_1, \ldots, \mathbf{n}_k) \mid \mathbf{t} \models \varphi[\mathbf{n}_1, \ldots, \mathbf{n}_k]\}$. By the Doner-Thatcher-Wright theorem there exists a tree automaton $\mathcal{M}$ (defined over G^k) such that $L(\mathcal{M}) = \mathcal{L}_\varphi$.

Hence, the theorem is proved if we can construct a RAG $\mathcal{R}$ which, on each tree $\mathbf{t}$, simulates $\mathcal{M}$ in parallel on all possible labelings of $\mathbf{t}$, outputting those labelings $(\mathbf{n}_1, \ldots, \mathbf{n}_k)$

[7] Immerman actually used the notation IND-DEPTH[$t(n)$].

$$
\begin{array}{ll}
U \to N & \mathit{result}(0) := \mathit{order}(1) \cup \{(1,0)\} \\
N \to NN & \mathit{descendants}(0) := \{(0)\} \cup \mathit{descendants}(1) \cup \mathit{descendants}(2) \\
 & \mathit{order}(0) := (\mathit{descendants}(1) \times \{(0)\}) \cup (\mathit{descendants}(2) \times \{(0)\}) \\
 & \qquad\qquad\quad \cup\, (\mathit{descendants}(1) \times \mathit{descendants}(2)) \\
 & \qquad\qquad\quad \cup\, \mathit{order}(1) \cup \mathit{order}(2) \\
N \to \ell & \mathit{descendants}(0) := \{(0),(1)\} \\
 & \mathit{order}(0) := \{(1,0)\}
\end{array}
$$

Figure 4: Computing a linear order on the nodes using a RAG.

such that $(\mathbf{t}, \mathbf{n}_1, \ldots, \mathbf{n}_k)$ is accepted by $\mathcal{M}$. Thereto, we use k-ary relation-valued synthesized attributes q for each state q of $\mathcal{M}$. The semantic rules are such that for each node $\mathbf{n}$, $(\mathbf{n}_1, \ldots, \mathbf{n}_k) \in q(\mathbf{n})$ iff $\mathcal{M}$ assumes state q on node $\mathbf{n}$ in its execution on $(\mathbf{t}, \mathbf{n}_1, \ldots, \mathbf{n}_k)$. The attribute *result* at the root is then defined as $\bigcup_{q\in F} q$, where F is the set of final states of $\mathcal{M}$. We omit the formal details. ∎

Like the proof of Theorem 3.1, the above proof involves the simulation in parallel of a tree automaton on different labelings of a tree. However here, the simulation is much more straightforward since we can simply parameterize the simulation, using relation-valued attributes. This was not possible in the case of BAGs (Theorem 3.1) which have only Boolean-valued attributes; the simulation there was much more intricate. In particular, while Theorem 6.1 states that synthesized attributes are enough for a RAG to express all of MSO, this is not the case for BAGs. Indeed, BAGs with only synthesized attributes are readily seen to be weaker than BAGs with both synthesized and inherited attributes.

We finally show that RAGs are strictly more powerful than MSO.

Call a grammar *unbounded* if it has infinitely many derivation trees. Clearly, if the grammar is bounded (i.e., not unbounded), then RAGs and MSO are equally powerful and collapse to first-order logic.

Proposition 6.2 *Over any unbounded grammar, RAGs can express Boolean (i.e., nullary) queries not definable in MSO.*

Proof. (Sketch) Consider an unbounded grammar G. There exists a sequence of productions $\bar{p} = p_1, \ldots, p_m$, a sequence of numbers $\bar{k} = k_1, \ldots, k_m$, and a non-terminal X, such that (i) X is the left-hand side of p_1 and occurs in position k_m of the right-hand side of p_m; (ii) for $i = 1, \ldots, m-1$ the symbol in position k_i of the right-hand side of p_i equals the non-terminal on the left-hand side of p_{i+1}; (iii) the left-hand sides of $p_1, \ldots, p_m$ are mutually distinct; and (iv) X is reachable from the start symbol.

Consider a derivation tree $\mathbf{t}$ of G. We say that a node $\mathbf{n}_1$ is an occurence of $(\bar{p}, \bar{k})$ in $\mathbf{t}$ if there exists a sequence of nodes $\mathbf{n}_2, \ldots, \mathbf{n}_m, \mathbf{n}_{m+1}$ such that for $i = 1, \ldots, m$, $\mathbf{n}_i$ is expanded with p_i and $\mathbf{n}_{i+1}$ is the k_i-th child of $\mathbf{n}_i$. We say that $\mathbf{n}_{m+1}$ is the tail of the occurence $\mathbf{n}_1$ of $(\bar{p}, \bar{k})$. Note that $\mathbf{n}_{m+1}$ is labeled with X. We call a sequence of nodes $\mathbf{n}_1, \ldots, \mathbf{n}_s$ a chain of occurences of $(\bar{p}, \bar{k})$ if for each $i = 1, \ldots, s-1$, $\mathbf{n}_i$ is an occurence of $(\bar{p}, \bar{k})$, $\mathbf{n}_{i+1}$ is the tail of the occurence $\mathbf{n}_i$, and $\mathbf{n}_s$ is not expanded with p_1.

Let $\mathcal{Q}$ be the Boolean query defined as follows: on every derivation tree $\mathbf{t}$, $\mathcal{Q}(\mathbf{t})$ is true if the length of the chain of occurences of $(\bar{p}, \bar{k})$ starting on the first X-labeled node in the preorder traversal of the tree is a power of two.

This query $\mathcal{Q}$ can be expressed by a RAG (details omitted).

We now show that $\mathcal{Q}$ is not expressible in MSO. Let φ be an MSO sentence. By the Doner-Thatcher-Wright theorem there exists a tree automaton $\mathcal{M} = (Q, \delta, F)$ accepting precisely the trees satisfying φ. Consider a tree $\mathbf{t}$ such that the length of the chain of occurences of $(\bar{p}, \bar{k})$ starting in the first X-labeled node in the preorder traversal of the tree is a power of two and is bigger than $|Q|+1$. There have to be two nodes $\mathbf{n}$ and $\mathbf{n}'$ of $\mathbf{t}$, such that $\mathbf{n}'$ is a descendant of $\mathbf{n}$, both are occurences of $(\bar{p}, \bar{k})$ in the chain, and $\bar{\delta}(\mathbf{t}_{\mathbf{n}}) = \bar{\delta}(\mathbf{t}_{\mathbf{n}'})$, where $\mathbf{t}_{\mathbf{x}}$ denotes the subtree of $\mathbf{t}$ with root $\mathbf{x}$. Let $\mathbf{t}'$ be the tree obtained from $\mathbf{t}$ by replacing subtree $\mathbf{t}_{\mathbf{n}'}$ by subtree $\mathbf{t}_{\mathbf{n}}$. Let c be the length of the chain of occurences, let $\mathbf{n}$ be the o_1-th occurence and $\mathbf{n}'$ be the o_2-th occurence in the chain. Then the chain of occurences of $(\bar{p}, \bar{k})$ starting in the first X-labeled node in the preorder traversal of $\mathbf{t}'$ has length $c + o_2 - o_1$. This is not a power of two because $c < c + o_2 - o_1 < 2c$, since $1 \leq o_1 < o_2 < c$. However, $\bar{\delta}(\mathbf{t}) = \bar{\delta}(\mathbf{t}')$, and thus $\mathbf{t}' \models \varphi$ if and only if $\mathbf{t} \models \varphi$. Hence, φ does not define $\mathcal{Q}$. ∎

As a corollary, RAGs can express more unary queries than BAGs.

7 Concluding remarks

BAGs as a language for expressing simple retrieval queries strike a reasonable balance between expressive power and complexity; on the one hand, they are as powerful as monadic second-order logic; on the other hand, they can be evaluated in linear time.

RAGs as a language for expressing general relational queries on structured documents offer more expressive power than BAGs, while remaining within polynomial-time complexity.

The logic PFP$[n]$, which we proved to be equivalent to RAGs, has a rather bizarre syntax, as it allows the iteration of a formula only when that formula is linearly bounded, which is not an obvious syntactic property. Actually, we do not know whether linear boundedness of first-order formulas over derivation trees of some fixed grammar is decidable. Over graphs the property can be shown undecidable by a reduction from validity; but over derivation trees (or equivalently Σ-trees, for some ranked alphabet Σ), first-order logic (even monadic second-order logic) is decidable [Tho97].

This problem of bizarre syntax can be avoided, however, by defining PFP$[n]$ in an alternative manner. Under this alternative, the iteration of *any* formula is allowed (so that the syntax is now obviously decidable). We then build into the semantics that the iteration is performed exactly n times, where n is the cardinality of the domain. It can be shown that our results continue to hold under this alternative view of PFP$[n]$.

As already mentioned, Theorem 3.1 was independently proved by Bloem and Engelfriet [BE97]. The other main result of their paper was that a binary query is MSO definable if and only if it is computable by a finite-state tree-walking automaton that can perform unary MSO tests. Two nodes $\mathbf{n}$ and $\mathbf{n}'$ are in the binary relation defined by the automaton if it can walk from $\mathbf{n}$ to $\mathbf{n}'$ starting from an initial state and ending in a final state.

The abbreviation RAG for relation-valued attribute grammar has also been used in the literature to denote a quite different concept, namely that of *relational attribute grammar* [DM93, CD88]. 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 relations among attributes. For instance, Deransart and Małuszynski [DM93] used relational attribute grammars to specify the valid proof trees of a logic program. As should be clear from the outset, the general notion of *relational* attribute grammar is not to be confused with our notion of RAG, being the instantiation of *standard* attribute grammars to relation-valued attributes and first-order definable functions.

References

[ACM93] S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *Proceedings 19th Conference on VLDB*, pages 73–84, 1993.

[ACM95] S. Abiteboul, S. Cluet, and T. Milo. A database interface for file updates. In M. Carey and D. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, volume 24:2 of *SIGMOD Record*, pages 386–397. ACM Press, 1995.

[BE97] 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.

[CD88] 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.

[DJL88] P. Deransart, M. Jourdan, and B. Lorho. *Attribute Grammars: Definition, Systems and Bibliography*, volume 323 of *Lecture Notes in Computer Science*. Springer, 1988.

[DM93] P. Deransart and J. Małuszynski. *A Grammatical View of Logic Programming*. MIT Press, 1993.

[Don70] J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1995.

[End72] H.B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[Gol95] C.F. Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1995.

[GS84] M. Gécseg and M. Steinby. *Tree Automata*. Akademia Kiadó, Budapest, 1984.

[GT87] 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.

[Imm89] N. Immerman. Expressibility and parallel complexity. *SIAM Journal on Computing*, 18:625–638, 1989.

[Knu68] 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.

[Knu82] D.E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, 1982.

[Mos74] Y.N. Moschovakis. *Elementary Induction on Abstract Structures*. North-Holland, 1974.

[NVdB97] 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, 1997.

[Tho97] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III. Springer, 1997.

[TW68] J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[Var89] M. Vardi. Automata theory for database theoreticians. *Proceedings of the 8th ACM Symposium on Principles of Database Systems*, pages 83-92. ACM Press, 1989.

[Woo95] 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.