# Expressiveness of structured document query languages based on attribute grammars[*]

Frank Neven[†] Jan Van den Bussche

Limburgs Universitair Centrum[‡]

**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. Further, we show that RAGs that only use synthesized attributes are strictly weaker than RAGs that use both synthesized and inherited attributes. We show that RAGs are more expressive than monadic second-order logic for queries of any arity.

[*]Preliminary versions of parts of this paper were presented at the 17th ACM Symposium on Principles of Database Systems (Seattle, June 1998), and at the 7th International Workshop on Foundations of Models and Languages for Data and Objects (Ostfriesland, October 1998).

[†]Post-doctoral researcher of the Fund for Scientific Research, Flanders.

[‡]Address: LUC, Dept. WNI, Universitaire Campus, B-3590 Diepenbeek, Belgium. E-mail: {fneven,vdbuss}@luc.ac.be.

Finally, we discuss relational attribute grammars in the context of BAGs and RAGs. We show that in the case of BAGs this does not increase the expressive power, while different semantics for relational RAGs capture the complexity classes NP, coNP and UP $\cap$ coUP.

## 1 Introduction

As originally proposed by Gonnet and Tompa [14], a structured document database can be naturally viewed as a derivation tree over a context-free grammar. In essence, this is also the view taken by SGML [13, 31].

The classical formalism of *attribute grammars*, introduced by Knuth [18], 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 [5]).

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 [1]. 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. We propose to use *Boolean-valued* attribute grammars (BAGs) to express such unary queries. 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 show that a unary query is expressible by a BAG if and only if it is definable in monadic second-order logic (MSO).[1] 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 [27, 6]) stating that a tree language is recognizable by a finite bottom-up 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. As a corollary of the proof, we obtain that every BAG is equivalent to a BAG that consists of one bottom-up pass followed by one top-down pass.

One can use BAGs also to express Boolean queries, and in this more restricted setting the equivalence between BAGs and MSO follows much more directly from the Doner-Thatcher-Wright Theorem. From this equivalence then follows a *bottom-up property* for Boolean BAG queries: every Boolean query expressible by a BAG is already expressible by a BAG using synthesized attributes only. This bottom-up property does *not* hold for BAGs expressing unary queries.

Having understood the expressive power of BAGs, we then turn to queries that result in relations, of arbitrary fixed arity, among the nodes of the tree. These queries are for example used when one wants to define "wrappers" that map relevant parts of the document into a relational database [1, 21, 25]. To this end, 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 Immerman [16] imply that these are precisely the queries computable in linear time on a parallel random access machine with polynomially many processors.

We also investigate whether the above-mentioned bottom-up property for Boolean BAG queries carries over to Boolean RAG queries; using tools from finite model theory we prove that it does not.

We complete the picture by showing that synthesized 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. Moreover, it turns out that each query

[1]We point out that this result was obtained independently by Bloem and Engelfriet [3].

defined by a monadic second-order logic formula can be expressed by a RAG that uses only synthesized attributes.

Finally, we consider Boolean-valued and relation-valued *relational* attribute grammars. Relational attribute grammars are introduced by Courcelle and Deransart [4]. 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. We discuss several natural semantics for relational BAGs and RAGs. We show that relational BAGs are entirely equivalent to standard BAGs. For RAGs, however, this is much less clear; under various semantics, relational RAGs capture complexity classes such as NP, coNP and UP $\cap$ coUP, whose relationship to the linear parallel time complexity class of standard RAGs is unknown.

The results obtained in this paper are summarized graphically in Figure 9.

This paper is further organized as follows. In Section 2, we introduce Boolean-valued and relation-valued attribute grammars as a query language. In Section 3, we characterize the expressive power of BAGs in terms of monadic second-order logic and establish a bottom-up property for Boolean BAG queries. In Section 4, we characterize the expressiveness of RAGs in terms of linear inductions of linear depth. Here, we also show that there is no bottom-up property for Boolean RAG queries, and we discuss the relationship between RAGs and MSO. Finally, in Section 5, we consider relational BAGs and RAGs. We present some concluding remarks in Section 6. Some technical proofs are moved to an appendix.

## 2 Attribute grammars as query languages

### 2.1 Data model

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. A *derivation tree of* $G$ is defined in the standard way (see, e.g., [15]).

Let $\mathbf{t}$ be a derivation tree of $G$ and let $\mathbf{n}_0$, $\mathbf{n}_1$, ..., $\mathbf{n}_n$ be nodes of $\mathbf{t}$ such that $\mathbf{n}_0$ has exactly the $n$ children $\mathbf{n}_1, \ldots, \mathbf{n}_n$:

$$\begin{tikzcd}[column sep=small] & \mathbf{n}_0 \arrow[dl] \arrow[dr] \arrow[d, phantom, "\cdots"] & \\ \mathbf{n}_1 & & \mathbf{n}_n . \end{tikzcd}$$

Let $p = X_0 \to X_1 \ldots X_n$ be a production. If the label of $\mathbf{n}_0$ is $X_0$, and $\mathbf{n}_i$ is labeled by $X_i$ for $i = 1, \ldots, n$, then we say that $\mathbf{n}_0$ is *derived* by $p$.

**Definition 2.1** The context-free grammar models the *schema* of the database. A *database instance* is a derivation tree of $G$.

**Definition 2.2** Let $k$ be a natural number. A *$k$-ary query* is a function $\mathcal{Q}$ that maps each derivation tree to a $k$-ary relation over its nodes. If $\mathcal{Q}$ is a nullary query, i.e., $k$ is zero, then we also say that $\mathcal{Q}$ is a *Boolean* query.

## 2.2 Attribute grammar formalism

We now define the concepts common to both Boolean-valued and relation-valued attribute grammars.

**Definition 2.3** An *attribute grammar vocabulary* has the form

$$(A, \mathrm{Syn}, \mathrm{Inh}, \mathrm{Att}),$$

where

- $A$ is a finite set 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 a synthesized and an inherited attribute of the same symbol, that terminal symbols do not have synthesized attributes, and that the start symbol does not have inherited attributes.

From now on we fix some attribute grammar vocabulary.

**Definition 2.4** Let $p = X_0 \to X_1 \ldots X_n$ be a production in $P$, and $a$ an attribute of $X_i$ for some $i \in \{0, \ldots, n\}$. 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 $i > 0$.

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

Figure 1: Example of a BAG.

## 2.3 Boolean-valued attribute grammars for unary and Boolean queries

**Definition 2.5** A BAG-*rule in the context* $(p, a, i)$, with $p = X_0 \to X_1 \dots X_n$, is an expression of the form

$$a(i) := \varphi,$$

where $\varphi$ 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 2.6** A *Boolean-valued attribute grammar (BAG)* $\mathcal{B}$ consists of an attribute grammar vocabulary, together with a mapping assigning to each context a BAG-rule in that context.

**Example 2.7** 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. ■

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 2.8** Let $\mathbf{t}$ be a derivation tree of $G$. A *valuation of* $\mathbf{t}$ is a function that maps pairs $(\mathbf{n}, a)$, where $\mathbf{n}$ is a node in $\mathbf{t}$ and $a$ is an attribute of the label of $\mathbf{n}$, to truth values (0 or 1).

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

**Definition 2.9** Let $\mathcal{B}$ be a BAG, and let $\mathbf{t}$ be a derivation tree. Let $a(i) := \varphi$ be the BAG-rule in context $(p, a, i)$. Let $\mathbf{n}_0$ be a node with children $\mathbf{n}_1, \dots, \mathbf{n}_n$ derived by $p$. Then the formula obtained from $\varphi$ by replacing each occurrence of a propositional symbol of the form $b(j)$ by the new propositional symbol $b(\mathbf{n}_j)$, is denoted by $\Delta(\mathcal{B}, \mathbf{t}, a, \mathbf{n}_i)$.

**Definition 2.10** Let $\mathcal{B}$ be a BAG and $\mathbf{t}$ a derivation tree. We define a sequence $(\mathcal{B}_l(\mathbf{t}))_{l \geq 0}$ of partial valuations as follows:

- $\mathcal{B}_0(\mathbf{t})$ is the empty valuation ($\mathcal{B}_0(\mathbf{t})$ is nowhere defined).
- $\mathcal{B}_l(\mathbf{t})$, for $l > 0$, is defined as the following extension of $\mathcal{B}_{l-1}(\mathbf{t})$. For every $a(\mathbf{n})$, if $\mathcal{B}_{l-1}(\mathbf{t})$ is defined on all propositional symbols that occur in $\Delta(\mathcal{B}, \mathbf{t}, a, \mathbf{n})$, then $\mathcal{B}_l(\mathbf{t})$ is defined on $a(\mathbf{n})$ and gets the truth value taken by $\Delta(\mathcal{B}, \mathbf{t}, a, \mathbf{n})$ under the valuation $\mathcal{B}_{l-1}(\mathbf{t})$.

If for every $\mathbf{t}$ there is an $l$ such that $\mathcal{B}_l(\mathbf{t})$ is a totally defined valuation of $\mathbf{t}$ (this implies that $\mathcal{B}_{l+1} = \mathcal{B}_l$), then we say that $\mathcal{B}$ is *non-circular.* From now on, we will only consider BAGs that are non-circular. (Non-circularity is well known to be decidable [18].) The valuation $\mathcal{B}(\mathbf{t})$ is then defined as $\mathcal{B}_l(\mathbf{t})$.

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., [5]). This is simply because only a constant number of attributes should be defined for every node. 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 $\mathbf{t}$.[2]

An arbitrary total valuation $v$ of $\mathbf{t}$ is said to *satisfy* $\mathcal{B}$ if $v(a(\mathbf{n}))$ equals the truth value taken by $\Delta(\mathcal{B}, \mathbf{t}, a, \mathbf{n})$ under $v$, for each attribute $a$ and node $\mathbf{n}$ of $\mathbf{t}$ such that $a$ is an attribute of the label of $\mathbf{n}$.

We shall make use of the following lemma:

**Lemma 2.11** *For each BAG $\mathcal{B}$ and derivation tree $\mathbf{t}$, $\mathcal{B}(\mathbf{t})$ is the only valuation that satisfies $\mathcal{B}$.*

[2]We use the RAM model of computation.

**Proof.** It follows immediately from the definitions that $\mathcal{B}(\mathbf{t})$ satisfies $\mathcal{B}$.

Suppose that $v$ satisfies $\mathcal{B}$. We now show by induction on $l$ that if $a(\mathbf{n})$ is defined in $\mathcal{B}_l(\mathbf{t})$ then $\mathcal{B}_l(\mathbf{t})(a(\mathbf{n})) = v(a(\mathbf{n}))$. This clearly holds for $l = 0$. Suppose $l > 0$ and $a(\mathbf{n})$ is defined in $\mathcal{B}_l(\mathbf{t})$. If $a(\mathbf{n})$ is already defined in $\mathcal{B}_{l-1}(\mathbf{t})$ then the claim holds by the inductive hypothesis. If $a(\mathbf{n})$ is not defined in $\mathcal{B}_{l-1}(\mathbf{t})$, then, by definition, the value $\mathcal{B}_l(\mathbf{t})(a(\mathbf{n}))$ equals the truth value of $\Delta(\mathcal{B}, \mathbf{t}, a, \mathbf{n})$ under the valuation $\mathcal{B}_{l-1}(\mathbf{t})$. By assumption $v(a(\mathbf{n}))$ equals the truth value of $\Delta(\mathcal{B}, \mathbf{t}, a, \mathbf{n})$ under the valuation $v$. By the inductive hypothesis we have that $\mathcal{B}_{l-1}(\mathbf{t})(b(\mathbf{m})) = v(b(\mathbf{m}))$, for all $b(\mathbf{m})$ that are defined in $\mathcal{B}_{l-1}(\mathbf{t})$. Hence, $\mathcal{B}_j(\mathbf{t})(a(\mathbf{n})) = v(a(\mathbf{n}))$.

By definition of $\mathcal{B}(\mathbf{t})$ the lemma now holds. ■

A BAG $\mathcal{B}$ can be used in a simple way to express unary (i.e., 1-ary) queries. Among the attributes in the vocabulary of $\mathcal{B}$, we designate some attribute *result*, and define:

**Definition 2.12** A BAG $\mathcal{B}$ expresses the unary query $\mathcal{Q}$ defined by

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

for every derivation tree $\mathbf{t}$.

**Example 2.13** 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 $\mathbf{n}$, $\mathit{even}(\mathbf{n})$ 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; $\mathit{x\_before}(\mathbf{n})$ 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 $\mathit{result}(\mathbf{n})$ as $\mathit{x\_before}(\mathbf{n}) \wedge \mathit{even}(\mathbf{n})$. 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. ■

A BAG can also be used to express Boolean (i.e., nullary) queries. Among the attributes of the start symbol, we designate some attribute *result*, and define:

$$\begin{tikzcd}
 & \begin{array}{c} \mathit{even} = 0 \\ \mathit{x\_before} = 0 \\ \mathit{result} = 0 \end{array} & \begin{array}{c} \mathit{even} = 1 \\ \mathit{x\_before} = 0 \\ \mathit{result} = 0 \end{array} & \begin{array}{c} \mathit{even} = 0 \\ \mathit{x\_before} = 0 \\ \mathit{result} = 0 \end{array} & \begin{array}{c} \mathit{even} = 1 \\ \mathit{x\_before} = 1 \\ \mathit{result} = 1 \end{array} & \begin{array}{c} \mathit{even} = 0 \\ \mathit{x\_before} = 1 \\ \mathit{result} = 0 \end{array} \\
U \arrow[r] & S \arrow[r] \arrow[d] & S \arrow[r] \arrow[d] & S \arrow[r] \arrow[d] & S \arrow[r] \arrow[d] & S \arrow[d] \\
 & B \;\; \mathit{is\_x} = 0 \arrow[d] & B \;\; \mathit{is\_x} = 0 \arrow[d] & B \;\; \mathit{is\_x} = 1 \arrow[d] & B \;\; \mathit{is\_x} = 0 \arrow[d] & B \;\; \mathit{is\_x} = 1 \arrow[d] \\
 & y & y & x & y & x
\end{tikzcd}$$

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

**Definition 2.14** A BAG $\mathcal{B}$ expresses the Boolean query $\mathcal{Q}$ defined by

$$\mathcal{Q}(\mathbf{t}) = \begin{cases} \text{true} & \text{if } \mathcal{B}(\mathbf{t})(\mathit{result}(\mathbf{r})) = 1; \\ \text{false} & \text{otherwise,} \end{cases}$$

for every derivation tree $\mathbf{t}$. Here $\mathbf{r}$ denotes the root of $\mathbf{t}$.

## 2.4 Relation-valued attribute grammars for relational queries

In this section, we generalize BAGs to *relation*-valued attribute grammars (RAGs). We start by giving an example.

**Example 2.15** 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 $S$ 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, intuitively, $d$ is a document, $f$ is the first paragraph of $d$, and $l$ is the last paragraph of $d$. More precisely, $d$, $f$, and $l$ are not actual parts of the derivation tree, but are just nodes corresponding to documents and paragraphs. The *result* relation is computed using the synthesized attributes *first* and *last* of $D$ and $L$; for every document node $\mathbf{n}$, *first*$(\mathbf{n})$ contains the first paragraph of that document, and *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

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

Figure 3: Example of a RAG. Technically, the rule bodies in this RAG are not strictly first-order formulas, but they can certainly be expressed in first-order logic.

$\mathbf{n}$, $\mathit{begin}(\mathbf{n})$ is true if $\mathbf{n}$ marks the beginning of a document, and $\mathit{end}(\mathbf{n})$ is true if $\mathbf{n}$ marks the end. In the rules, $\mathbf{1}$ is a constant that refers to the first child of the node where the rule is evaluated. So, in the definition of $\mathit{first}(0)$ for the production $L \to pL$, $\mathbf{1}$ refers to the $p$-labeled child. 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.

**Definition 2.16** To each attribute $a$ we associate an arity $r_a$ (a natural number). A RAG-rule in the context $(p, a, i)$, with $p = X_0 \to X_1 \dots X_n$, is an expression of the form

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

where $x_1, \dots, x_{r_a}$ are all the free variables occurring in $\varphi$. Further, $\varphi$ is a first-order logic formula over the vocabulary

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

where for each $j = 0, \ldots, n$, $b(j)$ is a relation symbol of arity $r_b$, and $\mathbf{j}$ is a constant symbol. A valuation of a derivation tree $\mathbf{t}$ is 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 $\mathbf{t}$. A RAG $\mathcal{R}$ consists of an attribute grammar vocabulary together with a mapping assigning to each context a RAG-rule in that context.

**Definition 2.17** Let $\mathcal{R}$ be a RAG, and let $\mathbf{t}$ be a derivation tree. Let $a(i) := \varphi$ be the RAG-rule in the context $(p, a, i)$. Let $\mathbf{n}_0$ be a node with children $\mathbf{n}_1, \ldots, \mathbf{n}_n$ derived by $p$. Then the formula obtained from $\varphi$ by replacing each occurrence of a relation symbol $b(j)$ by the relation symbol $b(\mathbf{n}_j)$, and by replacing each constant symbol $\mathbf{j}$ by the node $\mathbf{n}_j$, is denoted by $\Delta(\mathcal{R}, \mathbf{t}, a, \mathbf{n}_i)$.

**Definition 2.18** Let $\mathcal{R}$ be a RAG and $\mathbf{t}$ a derivation tree. We define a sequence $(\mathcal{R}_l(\mathbf{t}))_{l \geq 0}$ of partial valuations as follows:

- $\mathcal{R}_0(\mathbf{t})$ is the empty valuation ($\mathcal{R}_0(\mathbf{t})$ is nowhere defined).
- $\mathcal{R}_l(\mathbf{t})$, for $l > 0$, is defined as the following extension of $\mathcal{R}_{l-1}(\mathbf{t})$. For every $a(\mathbf{n})$, if $\mathcal{R}_{l-1}$ is defined on all relational symbols that occur in $\Delta(\mathcal{R}, \mathbf{t}, a, \mathbf{n})$, then $\mathcal{R}_l(\mathbf{t})$ is defined on $a(\mathbf{n})$ as the relation obtained by evaluating the FO-formula $\Delta(\mathcal{R}, \mathbf{t}, a, \mathbf{n})$ over the whole tree $\mathbf{t}$ where each relation symbol $b(\mathbf{m})$ in $\Delta(\mathcal{R}, \mathbf{t}, a, \mathbf{n})$ is interpreted by $\mathcal{R}_{l-1}(b(\mathbf{m}))$.

The valuation $\mathcal{R}(\mathbf{t})$ is then defined as $\mathcal{R}_l(\mathbf{t})$, where $l$ is such that $\mathcal{R}_l$ is a total valuation.

An arbitrary total valuation $v$ of $\mathbf{t}$ is said to *satisfy* $\mathcal{R}$ if $v(a(\mathbf{n}))$ equals the relation defined by the FO-formula $\Delta(\mathcal{R}, \mathbf{t}, a, \mathbf{n})$, where each relation symbol $b(\mathbf{n}_j)$ is interpreted by $v(b(\mathbf{n}_j))$. Analogous to Lemma 2.11, one can prove the following lemma:

**Lemma 2.19** *For each RAG $\mathcal{R}$ and derivation tree $\mathbf{t}$, $\mathcal{R}(\mathbf{t})$ is the only valuation that satisfies $\mathcal{R}$.*

A RAG can be used to express $k$-ary queries in a simple way. Among the attributes of the start symbol $U$ we designate some $k$-ary attribute *result*, and define:

**Definition 2.20** A RAG $\mathcal{R}$ expresses the query $\mathcal{Q}$ defined as follows: for any tree derivation $\mathbf{t}$, $\mathcal{Q}(\mathbf{t})$ equals the value of *result*$(\mathbf{r})$ in $\mathcal{R}(\mathbf{t})$, where $\mathbf{r}$ is the root of $\mathbf{t}$.

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

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

**Example 2.21** Another example of a RAG is depicted in Figure 4. The RAG expresses a binary (i.e., 2-ary) query. When applied to a tree, the query returns a linear order of the tree nodes corresponding to a postorder traversal [17] of the tree. This example can easily be generalized to arbitrary grammars. ■

As mentioned in the introduction RAGs can be seen as an abstract model for wrappers. These are tools that map relevant parts of the document at hand into, for instance, a relational database [1, 21, 25]. We give an example to illustrate this.

**Example 2.22** The grammar in Figure 5 models a list of publications. Each publication consists of a list of authors and a title. We now want a wrapper generating a binary relation consisting of all pairs $(a, t)$ such that $a$ is an author of a publication with title $t$. The RAG in Figure 5 expresses this transformation. Here, for every AuthorList node $\mathbf{n}$, $b(\mathbf{n})$ contains the set of authors in the author list associated to $\mathbf{n}$. Further, for every Pub node $\mathbf{n}$, $\mathit{result}(\mathbf{n})$ contains all pairs $(a, t)$ where $a$ is an author and $t$ is the title of the publication represented by $\mathbf{n}$.

Of course, the binary relation created by a real wrapper, as opposed to the abstraction of it by RAGs, would contain the actual string content (that is, actual names of authors and titles) rather than just the nodes in the document corresponding to them. ■

| | |
|---|---|
| PubList $\to$ Pub PubList | $\mathit{result}(0) := \mathit{result}(1) \cup \mathit{result}(2)$ |
| PubList $\to$ Pub | $\mathit{result}(0) := \mathit{result}(1)$ |
| Pub $\to$ AuthorList Title | $\mathit{result}(0) := b(1) \times \{\mathbf{2}\}$ |
| AuthorList $\to$ Author AuthorList | $b(0) := \{\mathbf{1}\} \cup b(2)$ |
| AuthorList $\to$ Author | $b(0) := \{\mathbf{1}\}$ |

Figure 5: RAGs as an abstraction of wrappers.

## 3 Expressive power of BAGs

In this section we characterize the expressive power of BAGs in terms of monadic second-order logic (MSO). As a corollary we obtain a bottom-up property for Boolean BAG queries. First we recall the definitions of tree automata and MSO.

## 3.1 Tree automata and MSO

### 3.1.1 Tree automata

In the theory of tree languages [11, 12, 30], trees are usually viewed as terms over a ranked alphabet. A ranked alphabet $\Sigma$ is a vocabulary of function symbols with associated arities. The set $\mathbf{T}_\Sigma$ of $\Sigma$-*trees* is inductively defined in the following manner: if $f$ is a function symbol in $\Sigma$ of arity $n$, and $\mathbf{t}_1$, …, $\mathbf{t}_n$ are $\Sigma$-trees, then also $f(\mathbf{t}_1, \dots, \mathbf{t}_n)$ is a $\Sigma$-tree (the base case of this definition is given by the constant symbols, i.e., the function symbols of arity 0). A $\Sigma$-tree $\mathbf{t}$ can be thought of as a labeled tree, the nodes of which are all subterms of $\mathbf{t}$ (including $\mathbf{t}$ itself), where each node of the form $f(\mathbf{t}_1, \dots, \mathbf{t}_n)$ is labeled $f$.

**Definition 3.1** A *(bottom-up deterministic) tree automaton* is a triple $\mathcal{M} = (Q, \delta, F)$, consisting of a finite set of states $Q$, a set $F \subseteq Q$ of final states, and a transition function $\delta$ mapping tuples of the form $(f, q_1, \dots, q_n)$, where $f \in \Sigma$ is of arity $n$ and $q_1, \dots, q_n \in Q$, to elements of $Q$. The function $\delta$ can be extended in the canonical manner to a mapping $\bar{\delta} : \mathbf{T}_\Sigma \to Q$. A $\Sigma$-tree $\mathbf{t}$ is *accepted* by $\mathcal{M}$ if $\bar{\delta}(\mathbf{t}) \in F$.

The set of all $\Sigma$-trees accepted by $\mathcal{M}$ is denoted by $L(\mathcal{M})$ (called the *tree language defined by* $\mathcal{M}$). A set of $\Sigma$-trees (a tree language) $\mathcal{T}$ is *recognizable* if there exists a tree automaton $\mathcal{M}$, such that $\mathcal{T} = L(\mathcal{M})$.

### 3.1.2 Monadic second-order logic

Let $r$ be the maximal arity of function symbols in the ranked alphabet $\Sigma$. A $\Sigma$-tree $\mathbf{t}$ can be naturally viewed as a finite structure (in the sense of mathematical logic [8, 9]) over the binary relation symbols $\{S_1, \ldots, S_r\}$ and the unary relation symbols $\{O_f \mid f \in \Sigma\}$. We denote the vocabulary associated with $\Sigma$ by $\tau_\Sigma$. The domain of $\mathbf{t}$, viewed as a structure, equals the set of nodes of $\mathbf{t}$. 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 $\mathbf{t}$. The set $O_f$ in $\mathbf{t}$ equals the set of $f$-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 standard first-order logic. A detailed exposition on this logic can be found in, e.g., Ebbinghaus and Flum's book [7].

**Example 3.2** Consider the following MSO-formula $\varphi(x, y)$:

$$(\forall X)\big((X(x) \wedge (\forall z_1)(\forall z_2)\big((X(z_1) \wedge \bigvee_{i=1}^{r} S_i(z_1, z_2)) \rightarrow X(z_2))\big) \rightarrow X(y)\big).$$

Note that $X$ is a set-variable; the other variables, in particular the free variables $x$ and $y$, are individual variables. For any tree $\mathbf{t}$ and two nodes $\mathbf{n}$ and $\mathbf{m}$ of $\mathbf{t}$, we have $\mathbf{t} \models \varphi[\mathbf{n}, \mathbf{m}]$ if and only if $\mathbf{n}$ is an ancestor of $\mathbf{m}$. Indeed, $x$ is an ancestor of $y$ iff every set of nodes that contains $x$ and is closed under the child relations, also contains $y$. ■

We denote the set of monadic second-order formulas over $\tau_\Sigma$ by MSO($\Sigma$). If $\Sigma$ is clear from the context we sometimes omit it and just write MSO.

Consider a $\Sigma$-tree $\mathbf{t}$ and a sequence $\mathbf{s}_1, \ldots, \mathbf{s}_k$ of sets of nodes of $\mathbf{t}$. We can view the tuple $(\mathbf{t}, \mathbf{s}_1, \ldots, \mathbf{s}_k)$ as a labeling of $\mathbf{t}$: a node $\mathbf{n}$ in $\mathbf{t}$ is labeled by $u_1 \ldots u_k$, where for $i = 1, \ldots, k$,

$$u_i := \begin{cases} 1 & \text{if } \mathbf{n} \in \mathbf{s}_i, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\Sigma^k$ be the ranked alphabet obtained from $\Sigma$ as follows: for each function symbol $f$ in $\Sigma$, we have for each $\bar{u} \in \{0,1\}^k$, the function symbol $f\bar{u}$ in $\Sigma^k$, of the same arity as $f$. So if $\mathbf{t}$ is a $\Sigma$-tree, then $(\mathbf{t}, \mathbf{s}_1, \ldots, \mathbf{s}_k)$ is a $\Sigma^k$-tree.

Let $\varphi(Z_1,\ldots,Z_k)$ be an MSO($\Sigma$)-formula where $Z_1,\ldots,Z_k$ are free set variables. By the above, the set

$$\mathcal{L}_\varphi := \{(\mathbf{t},\mathbf{s}_1,\ldots,\mathbf{s}_k) \mid \mathbf{t} \models \varphi[\mathbf{s}_1,\ldots,\mathbf{s}_k]\},$$

can be viewed as a set of $\Sigma^k$-trees.

**Definition 3.3** A $\Sigma^k$-tree language $\mathcal{T}$ is MSO($\Sigma$)-*definable* if there exists an MSO($\Sigma$)-formula $\varphi(Z_1,\ldots,Z_k)$ such that $\mathcal{T} = \mathcal{L}_\varphi$.

A standard theorem in the theory of logic and tree languages [28] now states:

**Theorem 3.4** [6, 27] *Let $k$ be a natural number. A $\Sigma^k$-tree language is MSO($\Sigma$)-definable if and only if it is recognizable.*

An important special case is when the tree language is defined by an MSO($\Sigma$)-*sentence* $\varphi$. Then the tree language defined by $\varphi$, $\mathcal{L}_\varphi := \{\mathbf{t} \mid \mathbf{t} \models \varphi\}$, is a set of $\Sigma$-trees, instead of a set of trees over an annotation of $\Sigma$.

### 3.1.3 Derivation trees as $\Sigma$-trees

We now link derivation trees with $\Sigma$-trees. We associate with the grammar $G$ a ranked alphabet $\Sigma_G$ as follows: every terminal symbol of $G$ is a constant symbol in $\Sigma$, and every production $p = X_0 \to X_1 \ldots X_n$ of $G$ is a function symbol in $\Sigma$ of arity $n$. A derivation tree of $G$ can now be naturally viewed as a $\Sigma_G$-tree.

In the previous section we defined MSO over $\Sigma$-trees. However, in the context of derivation trees, when we use the alphabet $\Sigma_G$, it is a bit more convenient to use the vocabulary $\tau_G = \{S_1,\ldots,S_r,(O_X)_{X\in N\cup T}\}$. Here $S_1$, $\ldots$, $S_r$ are defined as before, and for each $X \in N \cup T$, $O_X$ is the unary relation that specifies which nodes in the tree are labeled with $X$. The vocabularies $\tau_G$ and $\tau_{\Sigma_G}$ can easily be defined in terms of each other. Indeed, for each $X \in N$

$$O_X(x) \quad \equiv \quad \bigvee\{O_p(x) \mid p = X \to \ldots \in P\},$$

and for each $p = X_0 \to X_1 \ldots X_n \in P$

$$O_p(x_0) \quad \equiv \quad (\exists x_1)\ldots(\exists x_n)\left(\bigwedge_{i=1}^{n} S_i(x_0,x_i) \wedge \bigwedge_{i=0}^{n} O_{X_i}(x_i)\right).$$

Hence, w.l.o.g., we use the vocabulary $\tau_G$.

In the following, unless explicitly specified otherwise, if we say ‘tree’ we always mean ‘derivation tree of $G$’.

## 3.2 Main Theorem

We now show that the unary queries expressed by BAGs are exactly those definable in MSO. MSO can be used in the usual way to define unary queries.

**Definition 3.5** Let $\varphi(x)$ be an MSO($\Sigma$)-formula where $x$ is a free individual variable. Then $\varphi$ defines the unary query $\mathcal{Q}$ defined by

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

for every tree $\mathbf{t}$.

**Lemma 3.6** *Every query expressible by a BAG is definable in MSO.*

**Proof.** Let $\mathcal{B}$ be a BAG. We know from Lemma 2.11 that for each tree there exists only one valuation that satisfies $\mathcal{B}$. In MSO we can easily define this valuation. For each attribute $a$ we have a set variable $Z_a$. This variable will contain all the nodes for which the attribute $a$ is true in $\mathcal{B}(\mathbf{t})$. To this end, we associate a formula to each semantic rule in the following way. Consider a rule $a(i) := \varphi$ of $\mathcal{B}$ in the context $(p, a, i)$ for some production $p = X_0 \to X_1 \ldots X_n$ of $G$. Define the formula $\rho_{p,a,i}(z_a)$ as

$$\rho_{p,a,i}(z_a, (Z_b)_{b\in A}) :=$$

$$(\exists z_0)(\exists z_1)\ldots(\exists z_n)\left(\underbrace{\bigwedge_{j=0}^{n} O_{X_j}(z_j) \wedge \bigwedge_{j=1}^{n} S_j(z_0, z_j) \wedge z_a = z_i}_{(*)} \wedge \widehat{\varphi}\right),$$

where $\widehat{\varphi}$ is obtained from $\varphi$ by replacing each propositional symbol $b(j)$ occurring in $\varphi$ by $Z_b(z_j)$. Intuitively, formula $(*)$ states that

$$\begin{tikzcd} & z_0 \arrow[dl] \arrow[dr] \arrow[d, phantom, "\cdots"] & \\ z_1 & & z_n \end{tikzcd}$$

is derived by the production $p$. Formula $\hat{\varphi}$ states that $\varphi$ holds for $z_0, z_1, \ldots, z_n$, i.e., that $a(z_i)$ is true. We now define $\varphi_a$ as the following disjunction over all rules defining the attribute $a$:

$$\varphi_a(z_a, (Z_b)_{b\in A}) := \bigvee\{\rho_{p,a,i}(z_a, (Z_b)_{b\in A}) \mid (p, a, i) \text{ is a context}\}.$$

Define $\xi((Z_a)_{a\in A})$ as the formula

$$\bigwedge_{a\in A} (\forall z)(Z_a(z) \leftrightarrow \varphi_a(z,(Z_b)_{b\in A})).$$

Let $\mathbf{t}$ be a tree and for each $a \in A$ let $\mathbf{s}_a$ be a set of nodes of $\mathbf{t}$ such that $\mathbf{t} \models \xi[(\mathbf{s}_a)_{a\in A}]$. Then define the valuation $v$ as follows:

$$v(a(\mathbf{n})) := \begin{cases} 1 & \text{if } \mathbf{n} \in \mathbf{s}_a, \\ 0 & \text{otherwise.} \end{cases}$$

If follows from the definition of $\xi$ that $v$ satisfies $\mathcal{B}$. Since, according to Lemma 2.11, there exists only one valuation that satisfies $\mathcal{B}$, it follows that for each $\mathbf{t}$ there exists only one sequence of sets $(\mathbf{s}_a)_{a\in A}$ such that $\mathbf{t} \models \xi[(\mathbf{s}_a)_{a\in A}]$. Hence, the following formula defines the query expressed by $\mathcal{B}$:

$$\sigma(z) := (\exists Z_a)_{a\in A} \left(\xi((Z_a)_{a\in A}) \wedge Z_{result}(z)\right).$$

■

We now state and prove our first main result.

**Theorem 3.7** *A unary query is expressible by a BAG if and only if it is definable in MSO.*

**Proof.** The only-if direction is given by Lemma 3.6.

For the other direction, consider the unary query defined by the 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 $\varphi'(Z) := (\forall z)(Z(z) \leftrightarrow \varphi(z))$. The formula $\varphi'(Z)$ defines $Z$ as the set of nodes that are selected by $\varphi(z)$. By Theorem 3.4, the set $\mathcal{L}_{\varphi'} = \{(\mathbf{t},\mathbf{s}) \mid \mathbf{t} \models \varphi'[\mathbf{s}]\}$ is a recognizable $\Sigma^1_G$-tree language. Let $\mathcal{M} = (Q, F, \delta)$ be a tree automaton over $\Sigma^1_G$ recognizing this language. So for every derivation tree $\mathbf{t}$, $\mathcal{M}$ accepts exactly those pairs $(\mathbf{t},\mathbf{s})$ for which $\mathbf{t} \models \varphi'[\mathbf{s}]$. Note that we have constructed $\varphi'$ in such a way that for every tree $\mathbf{t}$ there is exactly one set $\mathbf{s}$ of nodes of $\mathbf{t}$ 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 $\mathbf{t}$, 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 $\varphi$.

**Idea.** 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 vocabulary $\Sigma^1_G$.) 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 $\mathbf{n}$ and each state $q$, *can-q*$(\mathbf{n})$ is true in $\mathcal{B}(\mathbf{t})$ iff $\mathcal{M}$ assumes state $q$ at $\mathbf{n}$ in its execution on some labeling of $\mathbf{t}$. Since the accepted labeling is unique, there is exactly one final state $q_F$ such that *can-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-q*$_F(\mathbf{r})$ is true, and for all other states $q$, *must-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-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-q*$'(\mathbf{p})$ is true. We show that $\mathcal{M}$ assumes state $q$ at $\mathbf{n}$ on the *accepted labeling* if and only if *must-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*.

**Construction.** Formally, the BAG $\mathcal{B}$ is constructed as follows. The set of attributes of $\mathcal{B}$ is defined as

$$A = \{\mathit{result}\} \cup \{\mathit{can\text{-}q} \mid q \in Q\} \cup \{\mathit{must\text{-}q} \mid q \in Q\},$$

where for each terminal $X$, $\mathrm{Inh}(X) = A$, and for each non-terminal $X$ that is not the start symbol, $\mathrm{Syn}(X) = \{\mathit{result}\} \cup \{\mathit{can\text{-}q} \mid q \in Q\}$ and $\mathrm{Inh}(X) = \{\mathit{must\text{-}q} \mid q \in Q\}$. For the start symbol $U$, $\mathrm{Syn}(U) = A$.

Let $p = X_0 \to X_1 \dots X_n$ be a production. We have the following semantic rules (as usual, the empty disjunction is false):

- For each $q \in Q$, and for every $j$ such that $X_j$ is a terminal add the rule

$$\mathit{can\text{-}q}(j) \;:=\; \begin{cases} \text{true} & \text{if } \delta(X_j 0) = q \text{ or } \delta(X_j 1) = q, \\ \text{false} & \text{otherwise.} \end{cases}$$

For each $q \in Q$ we have

$$\begin{aligned} \mathit{can}\text{-}q(0) \;:=\; & \bigvee\{\bigwedge_{i=1}^{n} \mathit{can}\text{-}q_i(i) \mid \\ & \qquad q_1, \ldots, q_n \in Q, \\ & \qquad \exists u \in \{0,1\} \text{ such that} \\ & \qquad \delta(pu, q_1, q_2, \ldots, q_n) = q\} \end{aligned}$$

- If $X_0$ is the start symbol then for each $q \in Q$ add the semantic rule

  $$\mathit{must}\text{-}q(0) \;:=\; \begin{cases} \mathit{can}\text{-}q(0) & \text{if } q \in F, \\ \text{false} & \text{otherwise.} \end{cases}$$

  From our assumption that the start symbol $U$ does not occur on the right-hand side of any production, we know that there is only one occurrence of $U$ in the tree and this is at the root.[3] So the second, top-down, pass will start at the root.

  For $j = 1, \ldots, n$, and $q \in Q$ add the rule

  $$\begin{aligned} \mathit{must}\text{-}q(j) \;:=\; & \bigvee\{(\mathit{must}\text{-}q'(0) \wedge \bigwedge_{i=1}^{n} \mathit{can}\text{-}q_i(i)) \mid \\ & \qquad q', q_1, \ldots, q_n \in Q, \\ & \qquad \exists u \in \{0,1\} \text{ such that} \\ & \qquad \delta(pu, q_1, q_2, \ldots, q_n) = q' \wedge q_j = q\} \end{aligned}$$

- Add the rule

  $$\begin{aligned} \mathit{result}(0) \;:=\; & \bigvee\{(\mathit{must}\text{-}q(0) \wedge \bigwedge_{i=1}^{n} \mathit{must}\text{-}q_i(i)) \mid \\ & \qquad q, q_1, \ldots, q_n \in Q, \text{ and} \\ & \qquad \delta(p1, q_1, q_2, \ldots, q_n) = q\}, \end{aligned}$$

  and and for every $j$ such that $X_j$ is a terminal add

  $$\mathit{result}(j) \;:=\; \bigvee\{\mathit{must}\text{-}q(j) \mid q \in Q, \delta(X_j 1) = q\}.$$

[3] One can get rid of this assumption by allowing the start symbol to have inherited attributes. The formalism of attribute grammars becomes much less elegant then, however. Hence our harmless technical assumption concerning the start symbol.

**Correctness.** We now establish the correctness of $\mathcal{B}$. Fix a derivation tree $\mathbf{t}$. Let $\ell$ be a labeling of the nodes of $\mathbf{t}$ with 0 or 1. We denote the corresponding labeled tree by $\ell(\mathbf{t})$. If $\mathbf{n}$ is a node of $\mathbf{t}$, then denote the subtree of $\mathbf{t}$ with root $\mathbf{n}$ by $\mathbf{t}(\mathbf{n})$. If $\ell'$ is a labeling of $\mathbf{t}(\mathbf{n})$ and $\ell$ is a labeling of $\mathbf{t}$, then $[\ell/\ell']$ is the following labeling of $\mathbf{t}$: for each node $\mathbf{m}$ of $\mathbf{t}$,

$$[\ell/\ell'](\mathbf{m}) = \begin{cases} \ell'(\mathbf{m}) & \text{if } \mathbf{m} \text{ is a node of } \mathbf{t}(\mathbf{n}), \\ \ell(\mathbf{m}) & \text{otherwise.} \end{cases}$$

The following lemma can be proved by a straightforward induction on the height of $\mathbf{n}$.

**Lemma 3.8** *For every node* $\mathbf{n}$ *of* $\mathbf{t}$*, can-*$q(\mathbf{n})$ *is true if and only if there exists a labeling* $\ell$ *of* $t$ *such that* $\mathcal{M}$ *assumes state* $q$ *at* $\mathbf{n}$ *in its execution on input* $\ell(\mathbf{t})$.

Since $\mathcal{M}$ is deterministic, if for a node $\mathbf{n}$, both *can-*$q_1(\mathbf{n})$ and *can-*$q_2(\mathbf{n})$ are true and $q_1 \neq q_2$, then there exist two different labelings $\ell_1$ and $\ell_2$ for $\mathbf{t}(\mathbf{n})$ such that $\mathcal{M}$ assumes state $q_1$ ($q_2$) at $\mathbf{n}$ in its execution on $\ell_1(\mathbf{t}(\mathbf{n}))$ ($\ell_2(\mathbf{t}(\mathbf{n}))$). There is exactly one labeling $\ell^*$ of $\mathbf{t}$ such that $\mathcal{M}$ accepts $\ell^*(\mathbf{t})$. In particular, $\mathcal{M}$ assumes a state $q \in F$ at the root $\mathbf{r}$ of $\mathbf{t}$ on $\ell^*(\mathbf{t})$. Hence, there is exactly one $q \in F$ such that *can-*$q(\mathbf{r})$ is true. Hence, by definition of the semantic rules for the attributes *must-*$q(\mathbf{r})$, only one *must-*$q(\mathbf{r})$ is true.

**Lemma 3.9** *For each* $\mathbf{n}$*, must-*$q(\mathbf{n})$ *is true if and only if* $\mathcal{M}$ *assumes state* $q$ *at* $\mathbf{n}$ *on* $\ell^*(\mathbf{t})$.

**Proof.** We show this by induction on the depth of $\mathbf{n}$. The base case, where $\mathbf{n}$ is $\mathbf{r}$, has just been treated.

Now, consider a node $\mathbf{n}_0$ with children $\mathbf{n}_1, \ldots, \mathbf{n}_n$ derived by the production $p = X_0 \to X_1 \ldots X_n$. Let $j \in \{1, \ldots, n\}$.

(i) Suppose must-$q(\mathbf{n}_j)$ is true. Then there exists a $u \in \{0,1\}$, and $q', q_1, \ldots, q_n \in Q$ with $q = q_j$ such that

$$\delta(pu, q_1, q_2, \ldots, q_n) = q', \tag{1}$$

and

$$\textit{must-}q'(\mathbf{n}_0) \wedge \bigwedge_{i=1}^{n} \textit{can-}q_i(\mathbf{n}_i)$$

is true. From $\bigwedge_{i=1}^{n} \mathit{can}\text{-}q_i(\mathbf{n}_i)$ it follows that for each $i = 1, \ldots, n$ there exists a labeling $\ell_i$ of $\mathbf{t}(\mathbf{n}_i)$, such that $\mathcal{M}$ assumes state $q_i$ at $\mathbf{n}_i$ on $\ell_i(\mathbf{t}(\mathbf{n}_i))$. Define the labeling $\ell$ of $\mathbf{t}(\mathbf{n}_0)$ as follows: $\ell(\mathbf{n}_0) = u$ and for $i = 1, \ldots, n$, if $\mathbf{n} \in \mathbf{t}(\mathbf{n}_i)$ then $\ell(\mathbf{n}) = \ell_i(\mathbf{n})$. From (1) it follows that $\mathcal{M}$ assumes state $q'$ at $\mathbf{n}_0$ on $\ell(\mathbf{t}(\mathbf{n}_0))$. From $\mathit{must}\text{-}q'(\mathbf{n}_0)$ it follows by the inductive hypothesis that $\mathcal{M}$ assumes state $q'$ at $\mathbf{n}_0$ on $\ell^*(\mathbf{t})$. Hence, the tree $[\ell^*/\ell](\mathbf{t})$ is accepted by $\mathcal{M}$. Since there is only one accepted labeling, it follows that $\ell$ is the restriction of $\ell^*$ to $\mathbf{t}(\mathbf{n}_0)$ and $\mathcal{M}$ assumes state $q$ at $\mathbf{n}_j$ on $\ell^*(\mathbf{t})$.

(ii) Suppose $\mathcal{M}$ assumes state $q$ at $\mathbf{n}_j$ on $\ell^*(\mathbf{t})$. Let $q'$ be the state that $\mathcal{M}$ assumes at $\mathbf{n}_0$ on $\ell^*(\mathbf{t})$. There have to be $q_1, \ldots, q_n \in Q$, with $q_j = q$, and a $u \in \{0, 1\}$ such that $\delta(pu, q_1, \ldots, q_n) = q'$, and such that $\mathcal{M}$ assumes state $q_i$ at $\mathbf{n}_i$ on $\ell^*(\mathbf{t})$, for $i = 1, \ldots, n$. By the inductive hypothesis $\mathit{must}\text{-}q'(\mathbf{n}_0)$ is true, and by Lemma 3.8, $\mathit{can}\text{-}q_i(\mathbf{n}_i)$ is true for each $i = 1, \ldots, n$. Hence, $\mathit{must}\text{-}q(\mathbf{n}_j)$ is true by definition. ■

We now show that for each node $\mathbf{n}$, result($\mathbf{n}$) is true in $\mathcal{B}(\mathbf{t})$ if and only if $\ell^*(\mathbf{n}) = 1$. Since for every derivation tree $\mathbf{t}$, $\ell^*(\mathbf{n}) = 1$ exactly if $\mathbf{t} \models \varphi[\mathbf{n}]$, the proof is complete.

1. Let $\mathbf{n}$ be a leaf node labeled with $X$.

   (i) Suppose $\mathit{result}(\mathbf{n})$ is true. Hence, there exists a state $q$ such that $\mathit{must}\text{-}q(\mathbf{n})$ is true and $\delta(X1) = q$. It follows from Lemma 3.9, that $\mathcal{M}$ assumes state $q$ at $\mathbf{n}$ on $\ell^*(\mathbf{t})$. It cannot be the case that $\delta(X0) = q$, because then $\mathcal{M}$ would accept two labelings of $\mathbf{t}$. Hence, $\ell^*(\mathbf{n}) = 1$.

   (ii) Suppose $\ell^*(\mathbf{n}) = 1$. Let $q$ be the state that $\mathcal{M}$ assumes at $\mathbf{n}$ on $\ell^*(\mathbf{t})$. Then $\delta(X1) = q$. Hence, $\mathit{result}(\mathbf{n})$ is true.

2. Let $\mathbf{n}$ be an interior node, with children $\mathbf{n}_1, \ldots, \mathbf{n}_n$, derived by production $p$.

   (i) Suppose $\mathit{result}(\mathbf{n})$ is true. Hence, there exist states $q_1, \ldots, q_n, q$ such that $\delta(p1, q_1, \ldots, q_n) = q$, and $\mathit{must}\text{-}q(\mathbf{n}) \wedge \bigwedge_{i=1}^{n} \mathit{must}\text{-}q_i(\mathbf{n}_i)$ is true. It follows from Lemma 3.9 that $\mathcal{M}$ assumes state $q$ at $\mathbf{n}$, and state $q_i$ at $\mathbf{n}_i$ on $\ell^*(\mathbf{t})$, for $i = 1, \ldots, n$. It cannot be the case that $\delta(p0, q_1, \ldots, q_n) = q$, because then $\mathcal{M}$ would accept two labelings of $\mathbf{t}$. Hence, $\ell^*(\mathbf{n}) = 1$.

(ii) Suppose $\ell^*(\mathbf{n}) = 1$. Let $q$ be the state that $\mathcal{M}$ assumes at $\mathbf{n}$ on $\ell^*(\mathbf{t})$, and for $i = 1, \ldots, n$, let $q_i$ be the state that $\mathcal{M}$ assumes at $\mathbf{n}_i$ on $\ell^*(\mathbf{t})$. Then $\delta(p1, q_1, \ldots, q_n) = q$. Hence, *result*($\mathbf{n}$) is true. ■

As a corollary of our proof of Theorem 3.7 we obtain a normal form for BAGs. The BAG described in the proof is special in two ways. First, it needs only positive formulas (involving only the connectives $\vee$ and $\wedge$, without $\neg$) in its semantic rules. Second, it can be evaluated on any tree by one bottom-up pass followed by one top-down pass. So we have the following:

**Corollary 3.10** *Every BAG is equivalent to one which uses only positive formulas in its semantic rules, and moreover which can be evaluated in two passes (more precisely, which is simply-2-pass* [5]*).*

Actually, part of the above corollary, that one can always find an equivalent BAG which uses only positive rules, can also quite easily be seen directly. Let $\mathcal{B}$ be BAG over the attribute grammar vocabulary $(A, \mathrm{Syn}, \mathrm{Inh}, \mathrm{Att})$. We construct an equivalent BAG $\mathcal{B}'$ over the attribute grammar vocabulary $(A', \mathrm{Syn}', \mathrm{Inh}', \mathrm{Att}')$ that does not use negation in its semantic rules in the following way. For each attribute $a$ we add an attribute $Na$ that becomes true if the attribute $a$ is false. Formally, $A' = A \cup \{Na \mid a \in A\}$, for each grammar symbol $X$,

$$\mathrm{Syn}'(X) = \mathrm{Syn}(X) \cup \{Na \mid a \in \mathrm{Syn}(X)\},$$

and

$$\mathrm{Inh}'(X) = \mathrm{Inh}(X) \cup \{Na \mid a \in \mathrm{Inh}(X)\}.$$

For each rule $a(i) := \varphi$ of $\mathcal{B}$ in context $(p, a, i)$, add the rule $a(i) := \tilde{\varphi}$ in context $(p, a, i)$ and the rule $Na(i) := \widetilde{\neg\varphi}$ in context $(p, Na, i)$ to $\mathcal{B}'$. The formula $\tilde{\psi}$, where $\psi = \varphi$ or $\psi = \neg\varphi$, is obtained from $\psi$ by transforming it into disjunctive normal form and then replacing each literal $\neg b(j)$ by $Nb(j)$.

**Example 3.11** The BAG in Example 2.7 uses negation to select all nodes on an even numbered position. In Figure 6 a BAG is depicted that retrieves those nodes without using negation. For clarity we replaced the attribute *Neven* by *odd*. ■

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

Figure 6: Example of a BAG without negation.

## 3.3 Bottom-up property for Boolean BAG queries

Another view of a BAG is that of a two-way version of finite bottom-up tree automata, alternative to the more classical two-way generalization of tree automata provided by Moriya [22]. The two-way generalization is provided by the two different types of attributes in a BAG: intuitively, synthesized attributes provide the bottom-up direction, and inherited attributes provide the top-down direction.

The following proposition relates BAGs and tree automata.

**Proposition 3.12** *For each tree automaton $\mathcal{M}$ there exists a BAG $\mathcal{B}$ such that for every derivation tree $\mathbf{t}$, $\mathcal{M}$ accepts $\mathbf{t}$ if and only if $\mathcal{B}$ accepts $\mathbf{t}$. This BAG uses only synthesized attributes.*

**Proof.** The execution of $\mathcal{M} = (Q, F, \delta)$ on $\mathbf{t}$ can easily be simulated by a BAG $\mathcal{B}$ having synthesized attributes $q$ for all states $q$ in $Q$. If $\mathbf{n}$ is a node of $\mathbf{t}$, then the attribute value $q(\mathbf{n})$ is true in $\mathcal{B}(\mathbf{t})$ iff $\mathcal{M}$ assumes state $q$ at $\mathbf{n}$ in its execution on $\mathbf{t}$. Let $p = X_0 \to X_1 \dots X_n$ be a production of $G$, $T(p) := \{j \in \{1, \dots, n\} \mid X_j \text{ is a terminal}\}$, and $N(p) := \{1, \dots, n\} - T(p)$. Add for each $q \in Q$ the semantic rule

$$q(0) :=$$
$$\bigvee \{ \bigwedge_{i \in N(p)} q_i(i) \mid \begin{array}{l} q_1, \dots, q_n \in Q \text{ such that } \delta(p, q_1, \dots, q_n) = q \\ \qquad\qquad \text{and } \delta(X_i) = q_i, \text{ for each } i \in T(p) \ \}. \end{array}$$

Finally, the attribute *result* of $U$ is defined by the rule $result(0) := \bigvee_{q \in F} q(0)$. ■

It now follows from Theorem 3.4 and Proposition 3.12, that every MSO-definable Boolean query is expressible by a synthesized BAG. This then leads to the following bottom-up property for Boolean BAG queries:

**Corollary 3.13** *For every BAG $\mathcal{B}$ there is a BAG $\mathcal{B}'$ having only synthesized attributes, such that $\mathcal{B}$ and $\mathcal{B}'$ express the same Boolean query.*

In the general case of arbitrary attribute grammars, where semantic rules can be arbitrary computable functions, it is well known that the use of inherited attributes can be simulated using synthesized attributes only [18]; we thus see that a similar phenomenon holds when semantic rules can only be propositional formulas.

Corollary 3.13 does not hold for BAGs expressing unary queries, as illustrated in the following example.

**Example 3.14** Consider again the grammar in Example 2.7. A query that can only be expressed with synthesized *and* inherited attributes is the one that retrieves all nodes, if both the first and the last letter of the string are $x$'s and retrieves no nodes otherwise. This query can not be expressed with only synthesized attributes. Indeed, every synthesized BAG already has to decide to select the last letter of the string without having visited the first letter, that is, without knowing whether the first letter carries an $x$. A same argument holds for BAGs having only inherited attributes. ■

## 4 Expressive power of RAGs

In this section we characterize RAGs as the queries defined by first-order inductions of linear depth, or, equivalently those computable in linear time on a parallel machine with polynomially many processors. We also show that, in contrast to BAGs, even for Boolean queries, synthesized RAGs are strictly less expressive than RAGs with both synthesized and inherited attributes. Hence, there is no bottom-up property for Boolean RAG queries. In the last subsection, we discuss the relationship between MSO and RAGs. First, we introduce the necessary logical definitions.

### 4.1 Fixpoint logic

See Ebbinghaus and Flum's book [7] for more background on the logics we are about to define.

### 4.1.1 Partial and least fixpoint logic

Fixpoint logic allows first-order logic formulas to be iterated. We will consider several kinds of fixpoint logics. Let $\varphi(z_1, \ldots, z_k, Z)$ be a first-order logic formula over the vocabulary $\tau_G$. The $z_i$'s are free individual variables, $Z$ is a $k$-ary relation variable that can be used in $\varphi$ in addition to the relation symbols provided by the vocabulary. On any tree $\mathbf{t}$, $\varphi$ defines the following relations obtained by iterating $\varphi$ starting with the empty relation for $Z$. Define

$$\begin{array}{rcl} \varphi^0(\mathbf{t}) & := & \emptyset; \\ \varphi^{i+1}(\mathbf{t}) & := & \{(\mathbf{n}_1, \ldots, \mathbf{n}_k) \mid \mathbf{t} \models \varphi[\mathbf{n}_1, \ldots, \mathbf{n}_k, \varphi^i(\mathbf{t})]\}. \end{array}$$

We say that $\varphi$ converges to a fixpoint on $\mathbf{t}$ if there exists an $n$ such that $\varphi^n(\mathbf{t}) = \varphi^{n+1}(\mathbf{t})$. We denote this fixpoint by $\varphi^\infty(\mathbf{t})$. If $\varphi$ does not reach a fixpoint on $\mathbf{t}$ we define $\varphi^\infty(\mathbf{t})$ as the empty set. We define *partial fixpoint logic (PFP)* as follows: formulas are constructed just as in first-order logic, with the addition that we also allow atomic formulas of the form $\mathrm{PFP}[\varphi, Z](z_1, \ldots, z_k)$, where $Z$ is $k$-ary and $\varphi(z_1, \ldots, z_k, Z)$ is a first-order logic formula. The semantics is as follows: for any tree $\mathbf{t}$, and nodes $\mathbf{n}_1$, ..., $\mathbf{n}_k$ of $\mathbf{t}$,

$$\mathbf{t} \models \mathrm{PFP}[\varphi, Z][\mathbf{n}_1, \ldots, \mathbf{n}_k] \quad \Leftrightarrow \quad (\mathbf{n}_1, \ldots, \mathbf{n}_k) \in \varphi^\infty(\mathbf{t}).$$

The formula $\varphi$ in $\mathrm{PFP}[\varphi, Z](z_1, \ldots, z_k)$ is called *positive* if every occurrence of the variable $Z$ occurs under an even number of negations. For such formulas the above described iteration process always reaches a fixpoint after a finite number of stages. Moreover, this fixpoint is also the least fixpoint of the operator defined by $\varphi$: over a tree $\mathbf{t}$, this operator maps $k$-ary relations $R$ over the domain of $\mathbf{t}$ to $k$-ary relations and is defined by

$$\varphi(R) := \{(\mathbf{n_1}, \ldots, \mathbf{n}_k) \mid \mathbf{t} \models \varphi[\mathbf{n_1}, \ldots, \mathbf{n}_k, R]\}.$$

We now define *least fixpoint logic (LFP)*, in the same way as PFP except that for each formula of the form $\mathrm{LFP}[\varphi, Z](z_1, \ldots, z_k)$, $\varphi$ has to be positive.

Note that our definitions of PFP and LFP differ from those in the literature: we do not allow nesting of fixpoints and we do not allow parameters in the formula constituting the fixpoint. However, since these can be dispensed with, our definitions are equivalent to the usual ones [7].

### 4.1.2 Fixpoints of linear depth

Consider a PFP-formula of the form $\mathrm{PFP}[\varphi, Z](z_1, \ldots, z_k)$. If there exist natural numbers $c$ and $d$ such that for every tree $\mathbf{t}$, $\varphi$ reaches its partial fixpoint after at most $c \cdot |\mathbf{t}| + d$ iterations, where $|\mathbf{t}|$ denotes the number of nodes in $\mathbf{t}$, then we say that $\varphi$ is *linearly bounded by the partial fixpoint semantics*. We define the logic PFP-LIN as the fragment of PFP where only partial fixpoints of linearly bounded formulas are allowed.

LFP-LIN is then the fragment of PFP-LIN that only allows formulas under the fixpoint operator that are both positive and linearly bounded.

### 4.1.3 Simultaneous fixpoint logic

Let $\varphi_1(\bar{z}_1, Z_1, \ldots, Z_k), \ldots, \varphi_k(\bar{z}_k, Z_1, \ldots, Z_k)$ be a system of first-order formulas, where for $j = 1, \ldots, k$, $Z_j$ is an $r_j$-ary relation variable. On a tree $\mathbf{t}$, consider for $j = 1, \ldots, k$, the stages defined by

$$\begin{array}{rcl} \varphi_j^0(\mathbf{t}) & := & \emptyset; \\ \varphi_j^{i+1}(\mathbf{t}) & := & \{(\mathbf{n}_1, \ldots, \mathbf{n}_{r_j}) \mid \mathbf{t} \models \varphi_j[\mathbf{n}_1, \ldots, \mathbf{n}_{r_j}, \varphi_1^i(\mathbf{t}), \ldots, \varphi_k^i(\mathbf{t})]\}. \end{array}$$

We say that this system reaches a *simultaneous fixpoint* on $\mathbf{t}$ if there exists an $n$ such that for all $j = 1, \ldots, k$, $\varphi_j^n(\mathbf{t}) = \varphi_j^{n+1}(\mathbf{t})$. We denote the relation defined by $\varphi_j$ in this fixpoint by $\varphi_j^\infty(\mathbf{t})$. If there does not exist a simultaneous fixpoint on $\mathbf{t}$, then $\varphi_j^\infty(\mathbf{t})$ is defined as the empty set. We now define *simultaneous partial fixpoint logic (S-PFP)* as follows: formulas are constructed just as in first-order logic, with the addition that we also allow formulas of the form $\text{S-PFP}_j[\varphi_1, \ldots, \varphi_k, Z_1, \ldots, Z_k](z_1, \ldots, z_r)$, where $Z_j$ is $r$-ary and $\varphi_i$ is a first-order formula for $i = 1, \ldots, k$. The semantics is defined as follows: for any tree $\mathbf{t}$, and nodes $\mathbf{n}_1$, $\ldots$, $\mathbf{n}_r$

$$\mathbf{t} \models \text{S-PFP}_j[\bar{\varphi}, \overline{Z}][\mathbf{n}_1, \ldots, \mathbf{n}_r] \quad \Leftrightarrow \quad (\mathbf{n}_1, \ldots, \mathbf{n}_r) \in \varphi_j^\infty(\mathbf{t}).$$

We say that the system of first-order formulas $\varphi_1, \ldots, \varphi_k$ is *linearly bounded by the simultaneous partial fixpoint semantics* if there exist natural numbers $c$ and $d$ such that for every derivation tree $\mathbf{t}$, the system of first-order formulas $\varphi_1, \ldots, \varphi_k$ reaches its simultaneous partial fixpoint after at most $c \cdot |\mathbf{t}| + d$ iterations. We define the logic S-PFP-LIN as the fragment of S-PFP where only simultaneous partial fixpoints of linearly bounded systems of first-order formulas are allowed.

The next proposition states that S-PFP-LIN is equivalent to PFP-LIN. In particular this means that mutual recursion can be replaced by simple recursion while preserving linearly boundedness. The proof is exactly as the proof of the Simultaneous Induction Lemma known from the theory of inductive definitions and finite model theory [23, 7].

**Proposition 4.1** *Every S-PFP-LIN-formula of the form*

$$\text{S-PFP}_j[\overline{\varphi}, \overline{Z}](\bar{z}),$$

*where* $j \in \{1, \ldots, k\}$*, is equivalent to a PFP-LIN-formula of the form*

$$(\exists \bar{u})(\text{PFP}[\psi, Z](\bar{z}\bar{u}));$$

## 4.2 Main Theorem

We now relate RAGs to PFP-LIN. PFP-LIN-formulas can express $k$-ary queries in the following way:

**Definition 4.2** Let $\varphi(x_1, \ldots, x_k)$ be a PFP-LIN-formula. Then $\varphi$ expresses the $k$-ary query $\mathcal{Q}$ defined by

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

for every tree $\mathbf{t}$.

**Theorem 4.3** *A query over derivation trees is expressible by a RAG if and only if it is definable in PFP-LIN.*

**Proof.** *Only if.* Let $\mathcal{R}$ be a RAG. We assume w.l.o.g. that no semantic rule contains a variable of $(z_a)_{a\in A}$, $z_0$, $z_1$, $z_2$, $\ldots$ . We define an S-PFP-LIN-formula that simulates $\mathcal{R}$. As induction variables of this system we have an[4] $(r_a + 1)$-ary relation variable $Z_a$ for each attribute $a$; $Z_a$ stands for the set of tuples $(\mathbf{n}, \mathbf{n}_1, \ldots, \mathbf{n}_{r_a})$, where $\mathbf{n}$ is a node labeled $X$ such that $a \in \text{Att}(X)$, and $(\mathbf{n}_1, \ldots, \mathbf{n}_{r_a})$ is a tuple in the currently computed value of $\mathcal{R}(\mathbf{t})(a(\mathbf{n}))$. For each attribute $a$ there is a formula $\varphi_a(z_a, x_1, \ldots, x_{r_a}, (Z_b)_{b\in A})$, defining

[4]Recall that $r_a$ is the arity of the attribute $a$.

the new value of $Z_a$ from the old values of the $Z_b$'s, built up as follows. Consider a rule $a(i) := \varphi(x_1, \ldots, x_{r_a})$ of $\mathcal{R}$ in the context $(p, a, i)$ for some production $p = X_0 \to X_1 \ldots X_n$ of $G$. The formula $\rho_{p,a,i}(z_a, x_1, \ldots, x_{r_a}, (Z_b)_{b \in A})$ is defined as

$$(\exists z_0)(\exists z_1) \ldots (\exists z_n) \left( \bigwedge_{j=0}^{n} O_{X_j}(z_j) \wedge \bigwedge_{j=1}^{n} S_j(z_0, z_j) \wedge z_a = z_i \wedge \widehat{\varphi} \right),$$

where $\widehat{\varphi}$ is obtained from $\varphi$ by replacing each occurrence of $b(j)(\bar{d})$ by $Z_b(z_j, \bar{d})$, and by replacing each occurrence of the constant symbol $\mathbf{k}$ by $z_k$. The formula $\varphi_a$ then is the disjunction over all rules defining the attribute $a$:

$$\begin{aligned} &\varphi_a(z_a, x_1 \ldots, x_{r_a}, (Z_b)_{b \in A}) := \\ &\qquad \bigvee \{\rho_{p,a,i}(z_a, x_1 \ldots, x_{r_a}, (Z_b)_{b \in A}) \mid (p, a, i) \text{ is a context}\}. \end{aligned}$$

Let $\sigma(\bar{z})$ be the formula

$$(\exists z) \left( O_U(z) \wedge \text{S-PFP}_1[\varphi_{result}, (\varphi_a)_{a \in A - \{result\}}, (Z_a)_{(a \in A)}](z, \bar{z}) \right).$$

Here $|\bar{z}|$ equals the arity of *result*. By an easy induction on $i$ one can now show that for any node $\mathbf{n}$ and attribute $a$, if $\mathcal{R}_i(\mathbf{t})$ is defined on $a(\mathbf{n})$ then for all nodes $\mathbf{m}_1, \ldots, \mathbf{m}_{r_a}$,

$$(\mathbf{n}, \mathbf{m}_1, \ldots, \mathbf{m}_{r_a}) \in \varphi_a^i(\mathbf{t}) \quad \Leftrightarrow \quad (\mathbf{m}_1, \ldots, \mathbf{m}_{r_a}) \in \mathcal{R}_i(\mathbf{t})(a(\mathbf{n})).$$

This implies that

$$(\mathbf{n}, \mathbf{m}_1, \ldots, \mathbf{m}_{r_a}) \in \varphi_a^{\infty}(\mathbf{t}) \quad \Leftrightarrow \quad (\mathbf{n}, \mathbf{m}_1, \ldots, \mathbf{m}_{r_a}) \in \mathcal{R}(\mathbf{t})(a(\mathbf{n})).$$

Further, for each tree $\mathbf{t}$, let $l_{\mathbf{t}}$ be the smallest integer such that $\mathcal{R}_{l_{\mathbf{t}}} = \mathcal{R}_{l_{\mathbf{t}}+1}$. Then, obviously, $l_{\mathbf{t}} \leq |A| \cdot |\mathbf{t}|$. Hence, the S-PFP-formula in $\sigma$ reaches its fixpoint after at most $|A| \cdot |\mathbf{t}|$ iterations. Proposition 4.1 now gives us the desired formula in PFP-LIN.

*If.* The crux of the proof is the simple observation that there is a RAG that computes 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 then be simulated in one preorder traversal of the tree, where the different stages are passed over as relational attribute values.

We now formally describe the RAG $\mathcal{R}$ that expresses the query defined by a PFP-LIN-formula. To compute the relations that make up a derivation tree we make use of the binary attributes $S'_1, \dots, S'_r$, where $r$ is the maximum width of any production in $P$, and the unary attributes $(O'_X)_{X \in N \cup T}$. These attributes are synthesized for non-terminals and inherited for terminals. They are defined by the following semantic rules. Consider the production $p = X_0 \to X_1 \dots X_n$. For $j =, 1 \dots, r$, define

$$S'_j(0) := \begin{cases} \bigcup_{i=1}^{n} S'_j(i) \cup \{(\mathbf{0}, \mathbf{j})\} & \text{if } j \leq n, \\ \bigcup_{i=1}^{n} S'_j(i) & \text{if } j > n. \end{cases}$$

For each $X \in N \cup T$, define

$$O'_X(0) := \begin{cases} \bigcup_{i=1}^{n} O'_X(i) \cup \{(\mathbf{0})\} & \text{if } X = X_0, \\ \bigcup_{i=1}^{n} O'_X(i) & \text{otherwise.} \end{cases}$$

For each $i$, such that $X_i$ is a terminal, and for each $j = 1, \dots, r$, define

$$S'_j(i) := \emptyset,$$

and for each $X \in N \cup T$ define

$$O'_X(i) := \begin{cases} \{(\mathbf{i})\} & \text{if } X = X_i, \\ \emptyset & \text{otherwise.} \end{cases}$$

The values of the relations $(S'_j)_{1 \leq j \leq r}$ and $(O'_X)_{X \in N \cup T}$ at the root then form the relational structure that represents the derivation tree. These values are now made available to the other nodes in the attributes $(S_j)_{1 \leq j \leq r}$ and $(O_X)_{X \in N \cup T}$. These attributes are synthesized for the start symbol $U$ and inherited for all other symbols, and are defined via the following rules. For every production of the form $U \to X_1 \dots X_n$, for every $j = 1, \dots, r$, and $X \in N \cup T$, define

$$S_j(0) := S'_j(0) \qquad \text{and} \qquad O_X(0) := O'_X(0).$$

For every production of the form $X_0 \to X_1 \ldots X_n$, where $X_0 \neq U$, and for every $j = 1, \ldots, r$, $i = 1, \ldots, n$, and $X \in N \cup T$, define

$$S_j(i) := S_j(0) \qquad \text{and} \qquad O_X(i) := O_X(0).$$

Let $\varphi$ be a PFP-LIN-formula. Then $\varphi$ is a first-order combination of formulas of the form $S_j(z_1, z_2)$, $O_X(z)$, and PFP$[\psi, Z](z_1, \ldots, z_k)$. Each relation $S_j$ and $O_X$ is already available at the root. Hence, it suffices to compute each subformula PFP$[\psi, Z](\bar{z})$ occurring in $\varphi$ in some attribute and make it available at the root.

Let $c$ and $d$ be numbers such that $\psi$ reaches its fixpoint after at most $c \cdot |\mathbf{t}| + d$ iterations on each tree $\mathbf{t}$. For any $i$, there exists a first-order formula $\psi^i(\bar{z}, Z)$ that defines $i$ stages of $\psi$ at once. Indeed, let $y_1, \ldots, y_k$ be variables that do not occur in $\psi$. Then, define $\psi^1(\bar{z}, Z)$ as $\psi(\bar{z}, Z)$, and for $i > 1$, $\psi^i(\bar{z}, Z)$ as the formula obtained from $\psi$ by replacing each atomic formula of the form $Z(\bar{d})$, by the formula $(\exists \bar{y})(\bar{y} = \bar{d} \wedge (\exists \bar{z})(\bar{z} = \bar{y} \wedge \psi^{i-1}(\bar{z}, Z)))$.

The RAG $\mathcal{R}^\psi$ evaluates the formula PFP$[\psi, Z](\bar{z})$ in the following way: First, $d$ stages of $\psi$ are evaluated at the root of the tree; this is achieved by evaluating the formula $\psi^d(\bar{z}, \emptyset)$. Then $\mathcal{R}^\psi$ makes a preorder traversal of the tree while evaluating $c$ stages of $\psi$, i.e., evaluating the formula $\psi^c$, at each node.

We now formally describe the RAG $\mathcal{R}^\psi$. It uses the $k$-ary attribute $Z$, which is synthesized for the start symbol and inherited for the other grammar symbols, and the $k$-ary attribute $Z'$, which is synthesized for the nonterminals and inherited for the terminals. The attributes $Z$ and $Z'$ are defined by the following semantic rules. Consider a production of the form $p = X_0 \to X_1 \ldots X_n$.

1. If $X_0 = U$ then add the rule

$$Z(0) := \{\bar{z} \mid \psi^{d+c}(\bar{z}, \emptyset)\}.$$

2. Further, define

$$Z(1) := \{\bar{z} \mid \psi_1^c(\bar{z})\},$$

   where $\psi_1^c$ is obtained from $\psi^c$ by replacing each occurrence of $O_X(z)$ or $S_h(z_1, z_2)$ by $O_X(1)(z)$ or $S_h(1)(z_1, z_2)$ respectively, and by replacing each occurrence of $Z(\bar{d})$ by $Z(0)(\bar{d})$.

3. For each $j$, such that $X_j$ is a terminal, define

$$Z'(j) := Z(j).$$

4. For each $j = 2, \ldots, n$, we have

$$Z(j) := \{\bar{z} \mid \psi_j^c(\bar{z})\},$$

where $\psi_j^c$ is obtained from $\psi^c$ by replacing each occurrence of $O_X(z)$ or $S_h(z_1, z_2)$ by $O_X(j)(z)$ or $S_h(j)(z_1, z_2)$ respectively, and by replacing each occurrence of $Z(\bar{d})$ by $Z'(j-1)(\bar{d})$.

5. Finally, define

$$Z'(0) := Z'(n).$$

Note that on every tree $\mathbf{t}$, the evaluation of $\mathcal{R}^\psi$ performs exactly $2 \cdot |\mathbf{t}|$ iterations. In each iteration exactly one attribute $Z$ or exactly one attribute $Z'$ is defined. For a tree $\mathbf{t}$, let for $i \geq 1$, $\alpha_i(\mathbf{t})$ ($\beta_i(\mathbf{t})$) be the number of $Z$ ($Z'$) attributes that are defined in $\mathcal{R}_i(\mathbf{t})$. The correctness of this construction now follows from the following lemma:

**Lemma 4.4** *Let* $\mathbf{t}$ *be a derivation tree, let* $\mathbf{n}$ *be a node of* $\mathbf{t}$ *and let* $a \in \{Z, Z'\}$. *If* $a(\mathbf{n})$ *is defined in* $\mathcal{R}_i^\psi(\mathbf{t})$ *but not in* $\mathcal{R}_{i-1}^\psi(\mathbf{t})$, *then*

$$\mathcal{R}_i^\psi(\mathbf{t})(a(\mathbf{n})) = \{\bar{\mathbf{n}} \mid \mathbf{t} \models \psi^{c \cdot \alpha_i(\mathbf{t}) + d}[\bar{\mathbf{n}}, \emptyset]\}.$$

This lemma can be proved by induction on the pair $(\alpha_i(\mathbf{t}), \beta_i(\mathbf{t}))$.

Hence, $\mathcal{R}_\psi(\mathbf{t})(Z'(\mathbf{r}))$ equals the relation defined by PFP$[\psi, Z](\bar{z})$, where $\mathbf{r}$ is the root of $\mathbf{t}$. ■

The logic PFP-LIN 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 $\Sigma$-trees, for some ranked alphabet $\Sigma$), satisfiability and validity of first-order logic (even monadic second-order logic) is decidable [6, 27, 28].

This problem of bizarre syntax can be avoided, however, by defining PFP-LIN in an alternative manner. Under this alternative, the iteration of *any* formula is allowed (so that the syntax is now trivially decidable). We then build into the semantics that the iteration is performed exactly $n$ times, where $n$ is the cardinality of the domain. It is not difficult to adapt the proof of Theorem 4.3 for this alternative view of PFP-LIN.

## 4.3 Complexity of RAGs

Immerman [16] showed that LFP-LIN captures the complexity class CRAM$[n]$ consisting of all queries computable in time $O(n)$ by a parallel machine with polynomially many processors.

**Theorem 4.5** [16] *LFP-LIN* = CRAM$[n]$ *on the class of all ordered finite structures.*

Using Theorem 4.3 and Theorem 4.5, we show the following:

**Corollary 4.6** *A query is expressible by a RAG if and only if it is computable in linear parallel time with polynomially many processors.*

**Proof.** In Lemma 4.7$(ii)$ we show that PFP-LIN = LFP-LIN on the class of all trees. Hence, by Theorem 4.5, we have that PFP-LIN = CRAM$[n]$ on the class of all ordered trees. By Theorem 4.3, it then suffices to show that PFP-LIN = CRAM$[n]$ on the class of all trees without a linear order. The order requirement in Theorem 4.5 is only needed to show that every CRAM$[n]$ program can indeed be simulated by an LFP-LIN formula. Hence, it readily follows that PFP-LIN $\subseteq$ CRAM$[n]$ on the class of all trees without a linear order. It remains to show the converse inclusion. Let $P$ be a CRAM$[n]$ program over trees with a linear order and let $\xi$ be the PFP-LIN formula simulating $P$. For expository purposes assume $\xi$ is of the form PFP$[\varphi, X](\bar{x})$. By Lemma 4.7$(i)$, there exists a PFP-LIN formula PFP$[\psi, Y](y_1, y_2)$ computing a linear order. We can not simply plug in the formula PFP$[\psi, Y](y_1, y_2)$ for each occurrence of $y_1 < y_2$ in $\xi$ because we do not allow nesting of fixpoints. However, we can use the following composition trick: we first compute the ordering and only then start iterating $\varphi$. That is, we just use the formula

$$\text{S-PFP}_1[\varphi', \psi, X, Y](\bar{x}), \qquad (*)$$

where $\varphi'$ is the formula $\rho_{\text{lin. ord.}}(Y) \wedge \hat{\varphi}$. Here, $\rho_{\text{lin. ord.}}(Y)$ is the first-order logic formula defining $Y$ as a *total* linear order, and $\hat{\varphi}$ is the formula obtained from $\varphi$ by replacing each occurrence of $y_1 < y_2$ by $Y(y_1, y_2)$. By definition of $\varphi'$, the iteration of $\varphi$ only starts when $Y$ is indeed a linear order. Further, $\varphi'$ is linearly bounded since both $\varphi$ and $\psi$ are. By Proposition 4.1, $(*)$ is equivalent to a PFP-LIN formula. ■

It remains to prove the following lemma.

**Lemma 4.7** *(i) There exists a PFP-LIN-formula that uniformly defines a total order on all trees.*

*(ii) PFP-LIN = LFP-LIN on the class of all trees.*

**Proof.** *(i)* Example 2.21 shows how an ordering of a binary tree can be obtained using a RAG. It is straightforward to generalize this construction to arbitrary derivation trees. By Theorem 4.3, this RAG is equivalent to a PFP-LIN-formula.

*(ii)* In Example 2.21, we saw how we can compute an ordering of a tree using a RAG. We can also compute this ordering directly in LFP-LIN. Hence, the equivalence of LFP-LIN and PFP-LIN on trees reduces to their equivalence on ordered trees (we can compose the computation of the ordering with other PFP constructs like in the proof of the previous theorem). The proof of the latter equivalence is similar to the proof of the known fact that LFP equals $\text{PFP}|_{\text{PTIME}}$ on ordered structures [7, Thm 7.4.14] (see also [2]). Here $\text{PFP}|_{\text{PTIME}}$ denotes the fragment of PFP, where every fixpoint is reached after at most a polynomial number of iterations. ■

## 4.4 No bottom-up property for RAGs

In this section we prove that *synthesized* RAGs, i.e., RAGs that only use synthesized attributes, are strictly less expressive than RAGs that can use both synthesized and inherited attributes.

For the rest of this section, let $G$ be the grammar $\{U \to LL, L \to L, L \to f\}$. Derivation trees of this grammar consists simply of two monadic trees concatenated at the root:

$$\begin{tikzcd} & U \arrow[dl] \arrow[dr] & \\ L \arrow[d] & & L \arrow[d] \\ \vdots \arrow[d] & & \vdots \arrow[d] \\ L & & L \end{tikzcd}$$

Let `equal_subtree` be the query that is true on $\mathtt{t}$ when the left subtree has the same number of nodes as the right subtree. We show that this query cannot be expressed by a synthesized RAG. However, it can be expressed by a RAG.

**Proposition 4.8** *The query* `equal_subtree` *is expressible by a RAG.*

**Proof.** By Theorem 4.3, it suffices to show that `equal_subtree` is definable in PFP-LIN:

$$\varphi := (\exists x)(\exists y)\big(x \neq y \wedge O_f(x) \wedge O_f(y) \wedge \mathrm{PFP}[\sigma, X](x, y)\big),$$

where

$$\begin{aligned} \sigma(x, y, X) := \ & (\exists z)\big(O_U(z) \wedge S_1(z, x) \wedge S_2(z, y)\big) \\ & \vee (\exists x')(\exists y')\big(X(x', y') \wedge S_1(x', x) \wedge S_1(y', y)\big). \end{aligned}$$

This formula maintains a binary relation $X$. In the first iteration of $\sigma$, the first node of the left subtree and the first node of the right subtree are put in $X$. In the following iterations, the next pair of corresponding nodes is added to $X$, provided it exists. Hence, $\sigma$ iterates at most $|\mathtt{t}|/2$ times on a tree $\mathtt{t}$, and thus belongs to PFP-LIN. The formula $\varphi$ then becomes true if both the last node of the left subtree and the last node of the right subtree belong to $X$. ■

In Appendix A we formally prove the next theorem.

**Theorem 4.9** *The query* `equal_subtree` *is not expressible by a synthesized RAG.*

## 4.5 RAGs versus MSO

We have characterized BAGs as the unary queries definable in MSO (Theorem 3.7), and RAGs as the queries (of arbitrary arity) definable in PFP-LIN (Theorem 4.3). It remains to compare these two formalisms with each other. We will show that synthesized RAGs are actually strictly more powerful than MSO.

MSO can be used in the standard way to define queries of any arity:

**Definition 4.10** Let $\varphi(x_1, \ldots, x_k)$ be an MSO-formula. Then $\varphi$ defines the $k$-ary query $\mathcal{Q}$ defined by

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

for every tree $\mathbf{t}$.

**Theorem 4.11** *Every $k$-ary query over derivation trees definable in MSO is expressible by a RAG using only synthesized attributes.*

**Proof.** Consider an MSO-formula $\varphi(z_1, \ldots, z_k)$. For any tree $\mathbf{t}$ and nodes $\mathbf{n}_1$, …, $\mathbf{n}_k$ 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 $i = 1, \ldots, k$,

$$u_i = \begin{cases} 1 & \text{if } \mathbf{n} = \mathbf{n}_i, \\ 0 & \text{otherwise.} \end{cases}$$

So $(\mathbf{t}, \mathbf{n}_1, \ldots, \mathbf{n}_k)$ is a $\Sigma_G^k$-tree as defined in Section 3.1. It is easy to write an MSO($\Sigma_G^k$)-sentence $\psi$ such that for every derivation tree $\mathbf{t}$ and nodes $\mathbf{n}_1$, …, $\mathbf{n}_k$ of $\mathbf{t}$:

$$(\mathbf{t}, \mathbf{n}_1, \ldots, \mathbf{n}_k) \models \psi \quad \Leftrightarrow \quad \mathbf{t} \models \varphi[\mathbf{n}_1, \ldots, \mathbf{n}_k].$$

Indeed, for $i = 1, \ldots, k$, define $J_i$ as the set of grammar symbols for which the $i$-th component is 1, i.e.,

$$J_i := \{X\bar{u} \mid X \in N \cup T, \bar{u} \in \{0,1\}^k \text{ and } u_i = 1\}.$$

Then $\psi$ is defined as

$$(\exists z_1) \ldots (\exists z_k) \\ \left( \left( \left( \bigvee_{X\bar{u} \in J_1} O_{X\bar{u}}(z_1) \right) \wedge \ldots \wedge \left( \bigvee_{X\bar{u} \in J_k} O_{X\bar{u}}(z_k) \right) \right) \to \varphi'(z_1, \ldots, z_k) \right),$$

where $\varphi'$ is the formula obtained by replacing each atomic formula $O_X(z)$ in $\varphi$ by $\bigvee_{\bar{u}\in\{0,1\}^k} O_{X\bar{u}}(z)$.

By Theorem 3.4 there exists a tree automaton $\mathcal{M}$ (defined over $\Sigma_G^k$) that accepts only those $(\mathbf{t}, \mathbf{n}_1, \ldots, \mathbf{n}_k)$ such that $(\mathbf{t}, \mathbf{n}_1, \ldots, \mathbf{n}_k) \models \psi$, where $\mathbf{t}$ is a derivation tree. Hence, the theorem is proved if we can construct a RAG $\mathcal{R}$ which, on each derivation tree $\mathbf{t}$, simulates $\mathcal{M}$ in parallel on all possible labelings of $\mathbf{t}$, returning those labelings $(\mathbf{n}_1, \ldots, \mathbf{n}_k)$ such that $(\mathbf{t}, \mathbf{n}_1, \ldots, \mathbf{n}_k)$ is accepted by $\mathcal{M}$. Thereto, we use a $k$-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 $\mathcal{M}$.

The formula $\mathrm{match}_{\bar{u}}(z_1, \ldots, z_k, y)$ defines the labelings $(z_1, \ldots, z_k)$ of the tree $\mathbf{t}$ such that node $y$ is labeled with $\bar{u}$:

$$\mathrm{match}_{\bar{u}}(z_1, \ldots, z_k, y) := \bigwedge_{u_i=1} z_i = y \wedge \bigwedge_{u_i=0} z_i \neq y.$$

Let $p = X_0 \to X_1 \ldots X_n$ be a production of $G$. Define $T(p) := \{i \in \{1, \ldots, n\} \mid X_i \text{ is a terminal}\}$ and $N(p) := \{1, \ldots, n\} - T(p)$. For each $q \in Q$ define the semantic rule

$$\begin{aligned} q(0) \ :=& \\ & \bigvee \{\mathrm{match}_{\bar{u}_0}(\bar{z}, 0) \wedge \bigwedge_{i\in N(p)} (\mathrm{match}_{\bar{u}_i}(\bar{z}, \mathbf{i}) \wedge q_i(\mathbf{i})(\bar{z})) \wedge \bigwedge_{i\in T(p)} \sigma_i(\bar{z}, \mathbf{i}) \mid \\ & \qquad \bar{u}_0, \ldots, \bar{u}_n \in \{0,1\}^k, \\ & \qquad q_1, \ldots, q_n \in Q, \\ & \qquad \delta(X_0\bar{u}_0 \to X_1\bar{u}_1 \ldots X_n\bar{u}_n, q_1, \ldots, q_n) = q\}, \end{aligned}$$

where $\sigma_i(\bar{z}, y)$ is the formula $\mathrm{match}_{\bar{u}_i}(\bar{z}, y)$ if $\delta(X\bar{u}_i) = q_i$ and is false otherwise.

The correctness of this construction now follows from the following lemma, which is easily proven by induction on the height of $\mathbf{n}$.

**Lemma 4.12** *Let* $\mathbf{t}$ *be a derivation tree. For each node* $\mathbf{n}$ *of* $\mathbf{t}$*,* $(\mathbf{m}_1, \ldots, \mathbf{m}_k) \in q(\mathbf{n})$ *iff* $\mathcal{M}$ *assumes state* $q$ *on node* $\mathbf{n}$ *in its execution on the labeled tree* $(\mathbf{t}, \mathbf{m}_1, \ldots, \mathbf{m}_k)$*.*

This concludes the proof. ■

The proof of the previous theorem, like the proof of Theorem 3.7, 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 3.7) which have only Boolean-valued attributes; the simulation there was much more intricate. In particular, while Theorem 4.11 states that synthesized attributes are enough for a RAG to express all of MSO, this is not the case for BAGs. Indeed as explained at the end of Section 3.3, BAGs with only synthesized attributes are weaker than MSO.

We finally show that synthesized RAGs are strictly more powerful than MSO. However, this only holds under the assumption that the underlying grammar can generate an infinite number of derivation trees.

**Definition 4.13** A grammar is *unbounded* if the number of its derivation trees is infinite.

Clearly, if the grammar is not unbounded, i.e., the number of derivation trees is finite, then RAGs and MSO are equally powerful because a query simply reduces to a case analysis.

**Theorem 4.14** *Over any unbounded grammar, synthesized RAGs can express Boolean queries not definable in MSO.*

**Proof.** 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 *occurrence* of $(\bar{p}, \bar{k})$ in $\mathbf{t}$ if there exists a sequence of nodes $\mathbf{n}_2$, ..., $\mathbf{n}_m$, $\mathbf{n}_{m+1}$ such that for $i = 1, \ldots, m$, $\mathbf{n}_i$ is derived 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 occurrence $\mathbf{n}_1$. 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 occurrences* of $(\bar{p}, \bar{k})$ if for each $i = 1, \ldots, s-1$, $\mathbf{n}_i$ is an occurrence, $\mathbf{n}_{i+1}$ is the tail of the occurrence

$\mathbf{n}_i$, and $\mathbf{n}_s$ is not derived with $p_1$. The length of the chain of occurrences $\mathbf{n}_1, \ldots, \mathbf{n}_s$ is $s$.

Let $\mathcal{Q}$ be the Boolean query defined as follows: on every derivation tree $\mathbf{t}$, $\mathcal{Q}(\mathbf{t})$ is true if there is a chain of occurrences starting on the first $X$-labeled node in the preorder traversal of the tree and its length is a power of two. Note that $\mathcal{Q}$ is true on any tree where there is no $X$-labeled node.

We now show that $\mathcal{Q}$ is not expressible in MSO. Let $\varphi$ be an MSO-sentence. By Theorem 3.4 there exists a tree automaton $\mathcal{M}' = (Q', \delta', F')$ accepting precisely the trees satisfying $\varphi$. Consider a tree $\mathbf{t}$ such that the length, which we denote by $c$, of the chain of occurrences 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 occurrences of $(\bar{p}, \bar{k})$ in the chain, and $\bar{\delta}(\mathbf{t}(\mathbf{n})) = \bar{\delta}(\mathbf{t}(\mathbf{n}'))$, where $\mathbf{t}(\mathbf{n})$ denotes the subtree of $\mathbf{t}$ with root $\mathbf{n}$. Let $\mathbf{n}$ be the $o_1$-th occurrence and $\mathbf{n}'$ be the $o_2$-th occurrence in the chain. Let $\mathbf{t}'$ be the tree obtained from $\mathbf{t}$ by replacing the subtree $\mathbf{t}(\mathbf{n}')$ by the subtree $\mathbf{t}(\mathbf{n})$. Then the chain of occurrences 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 $c < c + o_2 - o_1 < 2c$. However, $\bar{\delta}(\mathbf{t}) = \bar{\delta}(\mathbf{t}')$, and thus $\mathbf{t}' \models \varphi$ if and only if $\mathbf{t} \models \varphi$. Hence, $\varphi$ does not define $\mathcal{Q}$.

In Appendix B we formally show that $\mathcal{Q}$ is indeed expressible by a synthesized RAG. ■

As a corollary, we note:

**Corollary 4.15** *RAGs can express more unary queries than BAGs.*

## 5 Relational attribute grammars

Relational attribute grammars[5] are a generalization of standard attribute grammars introduced by Courcelle and Deransart [4]. In relational attribute grammars, the semantic rules no longer specify functions, computing attributes in terms of other attributes, but rather relations among attributes. Also there is no longer a distinction between synthesized and inherited attributes, and the values of the attributes are no longer uniquely determined

[5]Relational attribute grammars are not to be confused with the relation-valued standard attribute grammars (RAGs) of the previous section. In fact, in the present section, we will indeed consider relational versions of RAGs.

for every tree. We consider relational attribute grammars in the context of BAGs and RAGs, and discuss how they can express queries. We show that for BAGs this does not increase the expressive power, while in the case of RAGs the complexity classes NP, coNP and UP $\cap$ coUP are captured.

## 5.1 Relational BAGs

An attribute grammar vocabulary is now just a tuple $(A, \mathrm{Att})$, where $A$ is a finite set of attributes and Att is a function from $A$ to the powerset of $N \cup T$. A *relational BAG* $\mathcal{B}$ assigns to each production $p = X_0 \to X_1 \dots X_n$ a propositional formula $\varphi_p$ over the set of propositional symbols

$$\{a(j) \mid j \in \{0, \dots, n\}, a \in \mathrm{Att}(X_j)\}.$$

A *valuation* of a derivation tree $\mathbf{t}$ is a mapping that assigns a truth value to each $a(\mathbf{n})$, where $a \in A$, $\mathbf{n}$ is a node of $\mathbf{t}$, and $a$ is an attribute of the label of $\mathbf{n}$. Let $\mathbf{n}_0$ be a node of $\mathbf{t}$ with children $\mathbf{n}_1, \dots, \mathbf{n}_n$ derived by production $p$. Let $\varphi_p$ be the formula associated to $p$. Then define $\Delta(\mathcal{B}, \mathbf{t}, \mathbf{n}_0)$ as the formula obtained from $\varphi_p$ by replacing each propositional symbol of the form $b(j)$ by the new propositional symbol $b(\mathbf{n}_j)$. An arbitrary total valuation $v$ of $\mathbf{t}$ is said to *satisfy* $\mathcal{B}$ if $\Delta(\mathcal{B}, \mathbf{t}, \mathbf{n})$ is true under $v$ for every internal node $\mathbf{n}$.

A relational BAG can express unary queries in various ways. Let $\mathcal{Q}$ be a unary query and let $\mathcal{B}$ be a relational BAG. Designate among the attributes of $A$ an attribute *result*.

(*i*) $\mathcal{Q}$ is expressed *existentially* by $\mathcal{B}$ iff for every derivation tree $\mathbf{t}$

$$\mathcal{Q}(\mathbf{t}) = \{\mathbf{n} \mid \text{there exists a valuation } v \text{ of } \mathbf{t} \text{ that satisfies } \mathcal{B}, \text{ and } v(\mathit{result}(\mathbf{n})) \text{ is true}\};$$

(*ii*) $\mathcal{Q}$ is expressed *universally* by $\mathcal{B}$ iff for every derivation tree $\mathbf{t}$

$$\mathcal{Q}(\mathbf{t}) = \{\mathbf{n} \mid \text{for every valuation } v \text{ of } \mathbf{t} \text{ that satisfies } \mathcal{B}, v(\mathit{result}(\mathbf{n})) \text{ is true}\};$$

(*iii*) $\mathcal{Q}$ is expressed *implicitly* by $\mathcal{B}$ iff for every derivation tree $\mathbf{t}$ there exists exactly one valuation $v$ of $\mathbf{t}$ that satisfies $\mathcal{B}$, and $\mathbf{n} \in \mathcal{Q}(\mathbf{t})$ iff $v(\mathit{result}(\mathbf{n}))$ is true.

|  |  |
|---|---|
| $U \to S$ | $\neg x\_before(1)$ |
| $S \to BS$ | $(x\_before(2) \leftrightarrow is\_x(1) \vee x\_before(0))$ $\wedge\ (even(0) \leftrightarrow \neg even(2))$ $\wedge\ (result(0) \leftrightarrow even(0) \wedge x\_before(0))$ |
| $S \to B$ | $\neg even(0) \wedge \neg result(0)$ |
| $B \to x$ | $is\_x(0)$ |
| $B \to y$ | $\neg is\_x(0)$ |

Figure 7: Example of a relational BAG.

We denote the class of unary queries existentially (respectively, universally and implicitly) expressible by relational BAGs by ∃-BAG (respectively, ∀-BAG and IBAG).

**Example 5.1** In Figure 7 an example of a relational BAG is depicted. It expresses existentially, universally and implicitly the same query expressed by the BAG in Example 2.7. ■

The following theorem says that going from BAGs to relational BAGs does not increase the expressive power.

**Theorem 5.2** *BAG = ∃-BAG = ∀-BAG = IBAG.*

**Proof.** Clearly, by Lemma 2.11, BAG ⊆ ∃-BAG, BAG ⊆ ∀-BAG and BAG ⊆ IBAG. By using Theorem 3.7, we then only have to prove that every query in ∃-BAG, ∀-BAG and IBAG is definable in MSO.

1. Let $\mathcal{Q}$ be a query that is existentially expressed by the relational BAG $\mathcal{B}$. Then $\mathcal{Q}$ is defined by the MSO-formula $\psi(x) :=$

$$(\exists Z_a)_{a \in A}\Big(\Big(\bigwedge_{p = X_0 \to X_1 \dots X_n} (\forall x_0)\dots(\forall x_n)(p(x_0,\dots,x_n) \to \tilde{\varphi}_p)\Big) \wedge Z_{result}(x)\Big)$$

   where $A$ is the set of attributes of $\mathcal{B}$, $p(x_0,\dots,x_n)$ is the FO-formula that expresses that nodes $x_0,\dots,x_n$ are derived by production $p$, and $\tilde{\varphi}_p$ is the formula obtained from $\varphi_p$ by replacing each occurrence of $b(j)$ by $Z_b(x_j)$. Intuitively, the $Z_a$'s define valuations that satisfy $\mathcal{B}$.

2. Let $\mathcal{Q}$ be a query that is universally expressed by the relational BAG $\mathcal{B}$. Then $\mathcal{Q}$ is defined by the MSO-formula $\psi(x) :=$

$$(\forall Z_a)_{a\in A}\Big(\Big(\bigwedge_{p=X_0\to X_1\ldots X_n}(\forall x_0)\ldots(\forall x_n)(p(x_0,\ldots,x_n)\to\tilde{\varphi}_p)\Big) \to Z_{\text{result}}(x)\Big).$$

   Here $p(x_0,\ldots,x_n)$ and $\tilde{\varphi}_p$ are defined as in (1).

3. Let $\mathcal{Q}$ be a query that is implicitly expressed by the relational BAG $\mathcal{B}$. Then the MSO-formula that defines $\mathcal{Q}$ is the same as in (1). ■

## 5.2 Relational RAGs

Each attribute $a$ has an associated arity $r_a$. A *relational RAG* $\mathcal{R}$ associates to each production $p = X_0 \to X_1 \ldots X_n$ an FO-sentence $\varphi_p$ over the vocabulary

$$\bigcup_{j=0}^{n}\{a(j) \mid a \in \mathrm{Att}(X_j)\} \cup \{\mathbf{0},\mathbf{1},\ldots,\mathbf{n}\},$$

where for each $j = 0,\ldots,n$, $\mathbf{j}$ is a constant symbol and $a(j)$ is a relation symbol of arity $r_a$. A *valuation* of a derivation tree $\mathbf{t}$ is a mapping that assigns to each $a(\mathbf{n})$ an $r_a$-ary relation over the nodes of $\mathbf{t}$, where $a \in A$, $\mathbf{n}$ is a node of $\mathbf{t}$ and $a$ is an attribute of the label of $\mathbf{n}$. Let $\mathbf{n}_0$ be a node of $\mathbf{t}$ with children $\mathbf{n}_1,\ldots,\mathbf{n}_n$ derived by production $p$. Let $\varphi_p$ be the FO-sentence associated to $p$. Then define $\Delta(\mathcal{R},\mathbf{t},\mathbf{n}_0)$ as the FO-sentence obtained from $\varphi_p$ by replacing each occurrence of the relation symbol $b(j)$ by the relation symbol $b(\mathbf{n}_j)$ and by replacing each constant symbol $\mathbf{j}$ by the node $\mathbf{n}_j$. A valuation $v$ of $\mathbf{t}$ is said to *satisfy* $\mathcal{R}$ if $\Delta(\mathcal{R},\mathbf{t},\mathbf{n})$ evaluates to true for all $\mathbf{n}$ when each relation symbol $b(\mathbf{m})$ in $\Delta(\mathcal{R},\mathbf{t},\mathbf{n})$ is interpreted by $v$.

A relational RAG can express $k$-ary queries in various ways. Let $\mathcal{Q}$ be a $k$-query and let $\mathcal{R}$ be a relational RAG. Designate among the attributes of $A$ a $k$-ary attribute *result*. Let $\mathbf{r}$ denote the root of $\mathbf{t}$.

$(i)$ $\mathcal{Q}$ is expressed *existentially* by $\mathcal{R}$ iff for every derivation tree $\mathbf{t}$

$$\mathcal{Q}(\mathbf{t}) = \{(\mathbf{n}_1,\ldots,\mathbf{n}_k) \mid \text{there exists a valuation } v \text{ of } \mathbf{t} \text{ that satisfies } \mathcal{R}, \text{ and } (\mathbf{n}_1,\ldots,\mathbf{n}_k) \in v(\mathit{result}(\mathbf{r}))\}.$$

$$\begin{array}{ll}
U \to LL & (\forall x)(\exists! y)(C(1)(x) \to (C(2)(y) \wedge R(0)(x,y))) \\
 & \wedge\, (\forall y)(\exists! x)(C(2)(y) \to (C(1)(x) \wedge R(0)(x,y))) \\
 & \wedge\, \mathit{result}(0) \\
L \to L & (\forall x)(C(0)(x) \leftrightarrow x = \mathbf{0} \vee C(1)(x)) \\
L \to f & (\forall x)(C(0)(x) \leftrightarrow x = \mathbf{0} \vee x = \mathbf{1})
\end{array}$$

Figure 8: Example of a relational RAG.

*(ii)* $\mathcal{Q}$ is expressed *universally* by $\mathcal{R}$ iff for every derivation tree $\mathbf{t}$

$$\mathcal{Q}(\mathbf{t}) = \{(\mathbf{n}_1, \ldots, \mathbf{n}_k) \mid \text{for every valuation } v \text{ that satisfies } \mathcal{R} \\ (\mathbf{n}_1, \ldots, \mathbf{n}_k) \in v(\mathit{result}(\mathbf{r}))\}$$

*(iii)* $\mathcal{Q}$ is expressed *implicitly* by $\mathcal{R}$ iff for every derivation tree $\mathbf{t}$ there exists exactly one valuation $v$ of $\mathbf{t}$ that satisfies $\mathcal{R}$, and $(\mathbf{n}_1, \ldots, \mathbf{n}_k) \in \mathcal{Q}(\mathbf{t})$ iff $(\mathbf{n}_1, \ldots, \mathbf{n}_k) \in v(\mathit{result}(\mathbf{r}))$.

We denote the class of unary queries existentially (respectively, universally and implicitly) expressible by relational RAGs by ∃-RAG (respectively, ∀-RAG and IRAG).

**Example 5.3** In Figure 8 an example of a relational RAG $\mathcal{R}$ is depicted. This RAG expresses existentially the Boolean query which is true for a tree if the number of nodes in its left subtree equals the number of nodes in its right subtree. Let $\mathbf{t}$ be a tree, $\mathbf{r}$ the root of $\mathbf{t}$, $\mathbf{r}_1$ the left child and let $\mathbf{r}_2$ be the right child of the root. For any valuation $v$ of $\mathbf{t}$ that satisfies $\mathcal{R}$, $v(C(\mathbf{r}_1))$ contains the nodes of the left subtree, and $v(C(\mathbf{r}_2)$ contains the nodes of the right subtree. The sentence associated to the root can only be true under $v$ if $v(R(\mathbf{r}))$ contains a bijection between $v(C(\mathbf{r}_1))$ and $v(C(\mathbf{r}_2))$. ■

Clearly, any query expressible by a RAG is in ∃-RAG, ∀-RAG and IRAG. Indeed, let $a_1(i_1) := \varphi_1, \ldots, a_n(i_n) := \varphi_n$ be all the semantic rules in a RAG $\mathcal{R}$ associated to a production $p$. In the corresponding relational RAG, we just replace these by the single rule

$$(\forall \bar{x})(a_1(i_1)(\bar{x}) \leftrightarrow \varphi_1(\bar{x})) \wedge \ldots \wedge (\forall \bar{x})(a_n(i_n)(\bar{x}) \leftrightarrow \varphi_n(\bar{x})).$$

This is indeed a correct translation for all three discussed semantics since by Lemma 2.19 there exists only one valuation for each tree that satisfies $\mathcal{R}$.

In the following theorem, we characterize the 3 classes of relational RAG queried in terms of the complexity classes NP and UP (and their complements). NP is well known; UP is the class of problems decidable by a polynomial time non-deterministic Turing machine that is unambiguous, i.e., that has at most one accepting computation for every input [29, 24].[6] We obtain the following:

**Theorem 5.4** *1. $\exists$-RAG equals the class of queries in NP;*

*2. $\forall$-RAG equals the class of queries in coNP; and*

*3. IRAG equals the class of queries in UP $\cap$ coUP.*

**Proof.**

1. The containment $\exists$-RAG $\subseteq$ NP is clear. For the converse, we make use of Fagin's Theorem [10, 7], which states that the queries expressible in NP are exactly those that are definable in existential second-order logic (ESO). Every ESO-formula is of the form

   $$(\exists Z_1)\ldots(\exists Z_n)(\psi(\bar{x}, Z_1, \ldots, Z_n)),$$

   where the $Z_i$ are relation variables and $\psi$ is an FO-formula over the vocabulary expanded with the relation symbols $\{Z_1, \ldots, Z_n\}$.

   Consider the ESO-formula: $(\exists Z_1)\ldots(\exists Z_n)\psi(\bar{x}, \bar{Z})$. Like in the proof of Theorem 4.3, we can construct a RAG that computes all the relations that make up a derivation tree viewed as a relational structure. Add to this RAG the rule for the start symbol

   $$(\forall \bar{x})(\text{result}(0)(\bar{x}) \leftrightarrow \psi'(\bar{x})),$$

   where $\psi'$ is obtained from $\psi$ by replacing each $Z_i(\bar{y})$ by $Z_i(0)(\bar{y})$, and by replacing each relation of the vocabulary of the relational structure by its corresponding attribute. It then follows that this relational RAG expresses existentially the query defined by $(\exists Z_1)\ldots(\exists Z_n)\psi(\bar{x}, \bar{Z})$.

2. To prove that $\forall$-RAG are the queries computable in coNP, we make use of the complement of Fagin's Theorem: coNP = Universal second-order logic. The proof is then analogous to (1).

[6] A $k$-ary query $\mathcal{Q}$ belongs to a complexity class $\mathcal{C}$ if the decision problem $\{(\mathbf{t}, \mathbf{n}_1, \ldots, \mathbf{n}_k) \mid (\mathbf{n}_1, \ldots, \mathbf{n}_k) \in \mathcal{Q}(\mathbf{t})\}$ is in $\mathcal{C}$.

3. Clearly, IRAG $\subseteq$ UP $\cap$ coUP.

   Let $(\mathcal{Q}_1, \ldots, \mathcal{Q}_n)$ be a sequence of queries. We say that the sequence

   $$(\mathcal{Q}_1, \ldots, \mathcal{Q}_n)$$

   is *implicitly definable in FO* if there is an FO-sentence $\psi(Z_1, \ldots, Z_n)$ over the vocabulary of derivation trees augmented with $\{Z_1, \ldots, Z_n\}$ such that for every tree $\mathbf{t}$ the sequence $(\mathcal{Q}_1(\mathbf{t}), \ldots, \mathcal{Q}_n(\mathbf{t}))$ is the only sequence of relations $(R_1, \ldots, R_n)$ over $\mathbf{t}$ such that $\mathbf{t} \models \psi[R_1, \ldots, R_n]$.

   We write IMP(FO) to denote the collection of all queries $\mathcal{Q}$ such that $\mathcal{Q} = \mathcal{Q}_1$ for some sequence $(\mathcal{Q}_1, \ldots, \mathcal{Q}_n)$ of queries which is implicitly definable in FO.

   Analogously to (1) it can be shown that every query in IMP(FO) is expressible by a RAG. Kolaitis [20] proved that on every class of ordered structures, a query is definable in IMP(FO) if and only if it is computable in UP $\cap$ coUP. The trees we consider are not ordered. However, they can be ordered by a RAG, as we already saw in Lemma 4.7.

■

## 6 Concluding remarks

The results obtained in this paper are summarized in Figure 9. An arrow from a class of queries $C$ to a class of queries $C'$, means $C \subseteq C'$. A negated arrow from $C$ to $C'$, means there is a Boolean query in $C$ that is not in $C'$.

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.

As already mentioned, Theorem 3.7 was independently proved by Bloem and Engelfriet [3]. Actually, they did not investigate BAGs, but considered finite-valued attribute grammars. These are attribute grammars where the values for the attributes come from a fixed finite set. It is readily seen that they express the same unary queries as BAGs do. Bloem and Engelfriet,

$$\begin{tikzcd}
\exists\text{-RAG} = \text{NP} & \forall\text{-RAG} = \text{coNP} & \text{IRAG} = \text{UP} \cap \text{coUP} \\
& \text{RAG} = \text{PFP-LIN} = \text{CRAM}[n] \arrow[ul] \arrow[u] \arrow[ur] \arrow[d, shift left, "/" marking] & \\
& \text{synthesized RAG} \arrow[u, shift left] \arrow[d, shift left, "/" marking] & \\
& \text{BAG} = \exists\text{-BAG} = \forall\text{-BAG} = \text{IBAG} = \text{MSO} \arrow[u, shift left] \arrow[d, shift left, "/" marking, "\text{non-Boolean queries}"] & \\
& \text{synthesized BAG} \arrow[u, shift left] &
\end{tikzcd}$$

Figure 9: Summary of results.

however, did not study the expressiveness of attribute grammars as an abstract model of a query language. Their goal was proving the equivalence between two tree transformation languages. More precisely, their main result shows the equivalence between MSO tree transducers and two-stage attribute grammars that in the first stage compute a relabeling of the tree, by means of a finite valued attribute grammar, and in the second stage compute the output tree by means of tree-valued attributes.

## References

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

[2] S. Abiteboul and V. Vianu. Computing with first-order logic. *Journal of Computer and System Sciences*, 50(2):309–335, 1995.

[3] R. Bloem and J. Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *Journal of Computer and System Sciences*, 61(1):1–50, 2000.

[4] 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.

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

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

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

[8] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Undergraduate Texts in Mathematics. Springer-Verlag, second edition, 1994.

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

[10] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R.M. Karp, editor, *Complexity of Computation*, volume 7 of *SIAM-AMS Proceedings*, pages 43–73. 1974.

[11] F. Gécseg and M. Steinby. Tree languages. In Rozenberg and Salomaa [26], chapter 1.

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

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

[14] 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.

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

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

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

[18] 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.

[19] P.G. Kolaitis and M.Y. Vardi. Infinitary logics and 0-1 laws. *Information and Computation*, 98(2):258–294, 1992.

[20] Ph. Kolaitis. Implicit definability on finite structures and unambiguous computations. In *Proceedings 5th IEEE Symposium on Logic in Computer Science*, pages 168–180. IEEE Computer Society Press, 1990.

[21] G. Mecca, P. Atzeni, A. Masci, P. Merialdo, and G. Sindoni. The ARANEUS web-base management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume 27 of *ACM SIGMOD Record*, pages 544–546. ACM Press, 1998.

[22] E. Moriya. On two-way tree automata. *Information Processing Letters*, 50:117–121, 1994.

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

[24] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[25] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of the 11th International Conference on Data Engineering*, pages 251–260. IEEE Computer Society Press, 1995.

[26] G. Rozenberg and A. Salomaa, editors. *Handbook of formal languages*, volume 3. Springer, 1997.

[27] 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.

[28] W. Thomas. Languages, automata, and logic. In Rozenberg and Salomaa [26], chapter 7.

[29] Leslie G. Valiant. Relative complexity of checking and evaluating. *Information Processing Lettres*, 5(1):20–23, May 1976.

[30] 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.

[31] 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.

# Appendix A

In this appendix we prove Theorem 4.9. We start with the following definition.

**Definition 6.1** A *simple* RAG is a synthesized RAG over the attribute grammar vocabulary that has only the attribute $c$ for $L$ and only the zero-ary attribute *result* for $U$.

We focus attention on simple RAGs and show later that any synthesized RAG can be transformed into an equivalent simple one.

For any integers $n_1$ and $n_2$ greater than 1, let $\mathbf{t}(n_1, n_2)$ denote the tree which has a left subtree of length $n_1$ and a right subtree of length $n_2$. Let $\mathcal{R}$ be a simple RAG. In Lemma 6.3 we show that the values of $\mathcal{R}(\mathbf{t}(n_1, n_2))(c(\mathbf{n}_1))$ and $\mathcal{R}(\mathbf{t}(n_1, n_2))(c(\mathbf{n}_2))$, where $\mathbf{n}_1$ is the first child and $\mathbf{n}_2$ is the second child of the root, can be uniformly defined in PFP over a structure that, essentially, only contains an ordering of part of the domain of $\mathbf{t}(n_1, n_2)$. First, we need some definitions.

**Definition 6.2** Let $\tau_< = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, <\}$ be the vocabulary consisting of the constant symbols $\mathbf{0}$, $\mathbf{1}$ and $\mathbf{2}$, and the binary relation symbol $<$. Let $n_1$ and $n_2$ be two integers greater than 1.

1. Define $\mathcal{N}_1(n_1, n_2)$ as the $\tau_<$-structure with domain $\{1, \dots, n_1 + n_2 + 1\}$, where $\mathbf{0} = n_1 + n_2 + 1$, $\mathbf{1} = n_1$, $\mathbf{2} = n_1 + n_2$, and where $<$ is interpreted as the total order on $\{1, \dots, n_1\}$;

2. Define $\mathcal{N}_2(n_1, n_2)$ similarly as $\mathcal{N}_1(n_1, n_2)$, except that now $<$ is interpreted as the total order on $\{n_1 + 1, \dots, n_1 + n_2\}$.

Let $\xi(x_1, \dots, x_\ell)$ be a PFP-formula over the vocabulary $\tau_<$. We define $\xi(\mathcal{N}_i(n_1, n_2))$, for $i \in \{1, 2\}$, as the relation defined by $\xi$ on $\mathcal{N}_i(n_1, n_2)$, i.e., by

$$\{(\mathbf{n}_1, \dots, \mathbf{n}_\ell) \mid \mathcal{N}_i(n_1, n_2) \models \xi[\mathbf{n}_1, \dots, \mathbf{n}_\ell]\}.$$

**Lemma 6.3** *Let $\mathcal{R}$ be a simple RAG. There exists a PFP formula*

$$\xi(x_1, \dots, x_{r_c}),$$

*such that for all $n_1, n_2 > 1$*

$$\mathcal{R}(\mathbf{t}(n_1, n_2))(c(\mathbf{n}_1)) = \xi(\mathcal{N}_1(n_1, n_2)),$$

*and*

$$\mathcal{R}(\mathbf{t}(n_1, n_2))(c(\mathbf{n}_2)) = \xi(\mathcal{N}_2(n_1, n_2)),$$

*where* $\mathbf{n}_1$ *is the first child and* $\mathbf{n}_2$ *is the second child of the root of* $\mathbf{t}(n_1, n_2)$*.*

**Proof.** Let $c(0) := \varphi(\bar{x})$ be the rule in the context $(L \to f, c, 0)$, and let $c(0) := \psi(\bar{x})$ be the rule in the context $(L \to L, c, 0)$. Let $y_1$, $y_2$, $y_3$, $z_1$, $z_2$, $z_3$, $v_1$, …, $v_{r_c}$ be variables not occurring in $\varphi$ or $\psi$. Then define $\xi(\bar{x})$ as the formula

$$(\exists y_1)\,(\mathrm{root}(y_1) \wedge \mathrm{PFP}[\sigma, X](y_1, y_1, y_1, \bar{x}))\,.$$

Here $\mathrm{root}(y_1)$ is an FO-formula that defines the root of the tree, and

$$\sigma(y_1, y_2, y_3, x_1, \dots, x_{r_c}, X)$$

is the formula

$$\begin{aligned}&(y_1 = \mathrm{First} \wedge (y_2 = y_3 \to \varphi'(\bar{x})))\\ &\vee (\exists z_1)(\mathrm{Succ}(z_1) = y_1 \wedge (\exists z_2)(\exists z_3)(\exists \bar{v})(X(z_1, z_2, z_3, \bar{v}) \wedge (y_2 = y_3 \to \psi'(\bar{x}))),\end{aligned}$$

where First is the first element of $<$ and Succ is the successor function obtained from $<$; $\varphi'$ is obtained from $\varphi$ by replacing each occurrence of $\mathbf{1}$ by $y_1$ and each occurrence of $\mathbf{0}$ by $\mathrm{Succ}(y_1)$; $\psi'$ is obtained from $\psi$ by replacing each occurrence of $c(1)(\bar{d})$ by $X(z_1, z_1, z_1, \bar{d})$, each occurrence of $\mathbf{0}$ by $y_1$, and each occurrence of $\mathbf{1}$ by $z_1$. The variables $y_2$ and $y_3$ make sure that the relation $X$ is never empty. It might happen that $\varphi$ defines an empty relation; then the fixpoint would be empty. ∎

The next lemma is an immediate observation.

**Lemma 6.4** *Let* $\xi(x_1, \dots, x_\ell)$ *be a PFP-formula over* $\tau_<$*. For any* $n_1$ *and* $n_2$ *greater than 1, and* $i \in \{1, \dots, \ell\}$*:*

*(i) If* $\mathcal{N}_1(n_1, n_2) \models \xi[\mathbf{n}_1, \dots, \mathbf{n}_\ell]$ *and*

$$\mathbf{n}_i \in \{n_1 + 1, \dots, n_1 + n_2 - 1\} - \{\mathbf{n}_1, \dots, \mathbf{n}_{i-1}, \mathbf{n}_i, \dots \mathbf{n}_\ell\}$$

*then for all* $\mathbf{m} \in \{n_1 + 1, \dots, n_1 + n_2 - 1\} - \{\mathbf{n}_1, \dots, \mathbf{n}_\ell\}$*,*

$$\mathcal{N}_1(n_1, n_2) \models \xi[\mathbf{n}_1, \dots, \mathbf{n}_{i-1}, \mathbf{m}, \mathbf{n}_{i+1}, \dots, \mathbf{n}_\ell].$$

*(ii) If* $\mathcal{N}_2(n_1, n_2) \models \xi[\mathbf{n}_1, \ldots, \mathbf{n}_\ell]$ *and*

$$\mathbf{n}_i \in \{1, \ldots, n_1 - 1\} - \{\mathbf{n}_1, \ldots, \mathbf{n}_{i-1}, \mathbf{n}_i, \ldots \mathbf{n}_\ell\}$$

*then for all* $\mathbf{m} \in \{1, \ldots, n_1 - 1\} - \{\mathbf{n}_1, \ldots, \mathbf{n}_\ell\}$,

$$\mathcal{N}_2(n_1, n_2) \models \xi[\mathbf{n}_1, \ldots, \mathbf{n}_{i-1}, \mathbf{m}, \mathbf{n}_{i+1}, \ldots, \mathbf{n}_\ell].$$

**Proof.** This follows from the fact that PFP cannot distinguish between elements that are automorphic. Clearly, the transposition of any two elements $\mathbf{n}, \mathbf{n}' \in \{n_1 + 1, \ldots, n_1 + n_2 - 1\}$ is an automorphism of $\mathcal{N}_1(n_1, n_2)$, and the transposition of any two elements $\mathbf{n}, \mathbf{n}' \in \{1, \ldots, n_1 - 1\}$ is an automorphism of $\mathcal{N}_2(n_1, n_2)$. ■

**Definition 6.5** Let $\xi(x_1, \ldots, x_\ell)$ be a PFP-formula over the vocabulary $\tau_<$. Let $n_1$ and $n_2$ be two integers greater than 1. Let $\tau_{c,\ell}$ be the vocabulary $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, c_1, c_2\}$, consisting of the constant symbols $\mathbf{0}, \mathbf{1}, \mathbf{2}$ and the $\ell$-ary relation symbols $c_1$ and $c_2$. Define $\mathbf{t}(\xi, n_1, n_2)$ as the $\tau_{c,\ell}$-structure with domain $\{1, \ldots, n_1 + n_2 + 1\}$, where $\mathbf{0} = n_1 + n_2 + 1$, $\mathbf{1} = n_1$, $\mathbf{2} = n_1 + n_2$, $c_1 = \xi(\mathcal{N}_1(n_1, n_2))$, and $c_2 = \xi(\mathcal{N}_2(n_1, n_2))$.

We now show that, under certain assumptions, on the structures $\mathbf{t}(\xi, n_1, n_2)$ every FO-formula can be split into formulas that essentially speak only about the relation $c_1$ or only about the relation $c_2$, but not about both.

**Lemma 6.6** *Let $\xi$ be a PFP-formula with $\ell$ free variables. Assume there are FO-formulas $P_1(x)$ and $P_2(x)$ such that for all $n_1, n_2 > 1$,*

$$\mathbf{t}(\xi, n_1, n_2) \models P_1[\mathbf{n}] \quad \Leftrightarrow \quad \mathbf{n} \in \{1, \ldots, n_1 - 1\},$$

*and*

$$\mathbf{t}(\xi, n_1, n_2) \models P_2[\mathbf{n}] \quad \Leftrightarrow \quad \mathbf{n} \in \{n_1 + 1, \ldots, n_1 + n_2 - 1\}.$$

*For every FO-formula $\varphi(x_1, \ldots, x_k)$ over the vocabulary $\tau_{c,\ell}$ expanded with the unary relational symbols $P_1$ and $P_2$, there exists a disjunction $\psi(x_1, \ldots, x_k)$ of FO-formulas of the form*

$$\alpha(x_1, \ldots, x_k) \wedge \beta(x_1, \ldots, x_k) \wedge \bigwedge_{j=1}^{k} \omega_j(x_j),$$

*where $\alpha$ does neither contain $P_2$ nor $c_2$, $\beta$ does neither contain $P_1$ nor $c_1$, and each $\omega_j(x_j)$ is of the form $x_j = \mathbf{0}$, $x_j = \mathbf{1}$, $x_j = \mathbf{2}$, $P_1(x_j)$ or $P_2(x_j)$. For every $n_1, n_2 > 1$, and for every $\mathbf{n}_1, \ldots, \mathbf{n}_k \in \{1, \ldots, n_1 + n_2 + 1\}$ it holds that*

$$\mathbf{t}(\xi, n_1, n_2) \models \varphi[\mathbf{n}_1, \ldots, \mathbf{n}_k] \quad \Leftrightarrow \quad \mathbf{t}(\xi, n_1, n_2) \models \psi[\mathbf{n}_1, \ldots, \mathbf{n}_k].$$

**Proof.** The proof goes by induction on the structure of $\varphi$. We can assume w.l.o.g. that constants only appear in atomic formulas that are equalities.

1. If $\varphi$ equals $\mathbf{c} = \mathbf{c}'$, where $\mathbf{c}, \mathbf{c}' \in \{\mathbf{0}, \mathbf{1}, \mathbf{2}\}$ and $\mathbf{c} \neq \mathbf{c}'$, then $\psi$ is the empty disjunction.

2. If $\varphi$ equals $\mathbf{c} = \mathbf{c}$, where $\mathbf{c} \in \{\mathbf{0}, \mathbf{1}, \mathbf{2}\}$, then $\psi$ is (true $\wedge$ true).

3. Suppose $\varphi(x)$ is $x = \mathbf{c}$ or $\mathbf{c} = x$, where $\mathbf{c} \in \{\mathbf{0}, \mathbf{1}, \mathbf{2}\}$. Then $\psi$ is (true $\wedge$ true $\wedge$ $x = \mathbf{c}$).

4. Suppose $\varphi(x_1, x_2)$ is $x_1 = x_2$. We first introduce some notation. Define $D$ as the set of symbols $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, P_1, P_2\}$. Let $p$ be a natural number. For any $\bar{d} = d_1, \ldots, d_p \in D^p$, and variables $\bar{y} = y_1, \ldots, y_p$, define for $j = 1, \ldots, p$, the formula $\omega_{\bar{d},j}(\bar{y})$ as

$$\omega_{\bar{d},j}(\bar{y}) := \begin{cases} y_j = \mathbf{0} & \text{if } d_j = \mathbf{0}, \\ y_j = \mathbf{1} & \text{if } d_j = \mathbf{1}, \\ y_j = \mathbf{2} & \text{if } d_j = \mathbf{2}, \\ P_1(y_j) & \text{if } d_j = P_1, \\ P_2(y_j) & \text{if } d_j = P_2. \end{cases}$$

   Now define

$$\psi(x_1, x_2) := \bigvee_{d \in D} \left( x_1 = x_2 \wedge x_1 = x_2 \wedge \bigwedge_{j=1}^{2} \omega_{d,d,j}(x_1, x_2) \right).$$

5. Suppose $\varphi(x_1, \ldots, x_\ell)$ is $c_1(x_1, \ldots, x_\ell)$. Then define

$$\psi := \bigvee_{\bar{d} \in D^\ell} \left( c_1(x_1, \ldots, x_\ell) \wedge \text{true} \wedge \bigwedge_{j=1}^{\ell} \omega_{\bar{d},j}(\bar{x}) \right).$$

6. Suppose $\varphi(x_1, \ldots, x_\ell)$ is $c_2(x_1, \ldots, x_\ell)$. This is symmetric to (5).

7. Suppose $\varphi(x_1, \ldots, x_k)$ is $\varphi_1(y_1, \ldots, y_{k_1}) \vee \varphi_2(z_1, \ldots, z_{k_2})$. Here,

$$\{y_1, \ldots, y_{k_1}\} \cup \{z_1, \ldots, z_{k_1}\} = \{x_1, \ldots, x_k\}.$$

Let

$$\{u_1, \ldots, u_q\} = \{y_1, \ldots, y_{k_1}\} - \{z_1, \ldots, z_{k_1}\}$$

and let

$$\{v_1, \ldots, v_p\} = \{z_1, \ldots, z_{k_1}\} - \{y_1, \ldots, y_{k_1}\}.$$

By the inductive hypothesis, there exists a $\psi_1$ equivalent to $\varphi_1$ of the form

$$\bigvee_{i=1}^{n} \left( \alpha_i^1 \wedge \beta_i^1 \wedge \bigwedge_{j=1}^{k_1} \omega_{i,j}^1 \right),$$

and a $\psi_2$ equivalent to $\varphi_2$ of the form

$$\bigvee_{i=1}^{m} \left( \alpha_i^2 \wedge \beta_i^2 \wedge \bigwedge_{j=1}^{k_2} \omega_{i,j}^2 \right).$$

The formula $\psi$ is obtained from $\psi_1$ and $\psi_2$, by replacing every disjunct $\alpha_i^1 \wedge \beta_i^1 \wedge \bigwedge_{j=1}^{k_1} \omega_{i,j}^1$ in $\psi_1$ by

$$\bigvee_{\bar{d} \in D^p} \left( \alpha_i^1 \wedge \beta_i^1 \wedge \bigwedge_{j=1}^{k_1} \omega_{i,j}^1 \wedge \bigwedge_{j=1}^{p} \omega_{\bar{d},j}(\bar{v}) \right),$$

and by replacing every disjunct $\alpha_i^2 \wedge \beta_i^2 \wedge \bigwedge_{j=1}^{k_2} \omega_{i,j}^2$ in $\psi_2$ by

$$\bigvee_{\bar{d} \in D^q} \left( \alpha_i^2 \wedge \beta_i^2 \wedge \bigwedge_{j=1}^{k_2} \omega_{i,j}^1 \wedge \bigwedge_{j=1}^{q} \omega_{\bar{d},j}(\bar{u}) \right).$$

8. Suppose $\varphi(x_1, \ldots, x_k)$ is $\neg\varphi'(x_1, \ldots, x_k)$. By the inductive hypothesis, there is a $\psi'$ equivalent to $\varphi'$ of the form

$$\bigvee_{i=1}^{n} \left( \alpha_i' \wedge \beta_i' \wedge \bigwedge_{j=1}^{k} \omega_{i,j}' \right).$$

Then $\neg\psi'$ is equivalent to

$$\bigwedge_{i=1}^{n} \left( \neg\alpha'_i \vee \neg\beta'_i \vee \bigvee_{j=1}^{k} \neg\omega'_{i,j} \right).$$

We now transform this formula to an equivalent one in the right form. Replace each $\neg\omega'_{i,j}$ of the form

(a) $\neg(x_j = \mathbf{0})$ by $x_j = \mathbf{1} \vee x_j = \mathbf{2} \vee P_1(x_j) \vee P_2(x_j)$;

(b) $\neg(x_j = \mathbf{1})$ by $x_j = \mathbf{0} \vee x_j = \mathbf{2} \vee P_1(x_j) \vee P_2(x_j)$;

(c) $\neg(x_j = \mathbf{2})$ by $x_j = \mathbf{0} \vee x_j = \mathbf{1} \vee P_1(x_j) \vee P_2(x_j)$;

(d) $\neg P_1(x_j)$ by $x_j = \mathbf{0} \vee x_j = \mathbf{1} \vee x_j = \mathbf{2} \vee P_2(x_j)$;

(e) $\neg P_2(x_j)$ by $x_j = \mathbf{0} \vee x_j = \mathbf{1} \vee x_j = \mathbf{2} \vee P_1(x_j)$.

Put the resulting formula in disjunctive normal form (here the literals are the formulas $\neg\alpha'_i$, $\neg\beta'_i$, $z = \mathbf{0}$, $z = \mathbf{1}$, $z = \mathbf{2}$, $P_1(z)$ and $P_2(z)$). Each disjunct now looks like

$$\neg\alpha_{i_1} \wedge \ldots \wedge \neg\alpha_{i_s} \wedge \neg\beta_{i_1} \wedge \ldots \wedge \neg\beta_{i_r} \wedge \delta_1 \wedge \ldots \wedge \delta_b,$$

where each $\delta$ is of the form $z = \mathbf{0}$, $z = \mathbf{1}$, $z = \mathbf{2}$, $P_1(z)$ or $P_2(z)$. The disjunct is discarded if there are two different $\delta's$ for the same variable (e.g., one is $z = \mathbf{0}$ and the other is $P_1(z)$). Otherwise, define $\alpha$ as the formula $\neg\alpha_{i_1} \wedge \ldots \wedge \neg\alpha_{i_s}$, and define $\beta$ as the formula $\neg\beta_{i_1} \wedge \ldots \wedge \neg\beta_{i_r}$. Let $y_1, \ldots, y_g$ be the variables in $\{x_1, \ldots, x_k\}$ for which there is no $\delta$ in the disjunct. Then replace this disjunct by

$$\bigvee_{\bar{d} \in D^g} \left( \alpha \wedge \beta \wedge \bigwedge_{j=1}^{b} \delta_j \wedge \bigwedge_{j=1}^{g} \omega_{\bar{d},j}(\bar{y}) \right).$$

9. Suppose $\varphi(x_1, \ldots, x_k)$ is $(\exists x_{k+1})\varphi'(x_1, \ldots, x_k, x_{k+1})$. By the inductive hypothesis, there is a $\psi'$ equivalent to $\varphi'$ of the form

$$\bigvee_{i=1}^{m} \left( \alpha'_i \wedge \beta'_i \wedge \bigwedge_{j=1}^{k+1} \omega'_{i,j} \right).$$

Then $\varphi$ is equivalent to

$$\bigvee_{i=1}^{m}(\exists x_{k+1})\left(\alpha'_i \wedge \beta'_i \wedge \bigwedge_{j=1}^{k+1}\omega'_{i,j}\right).$$

This is then equivalent to

$$\bigvee_{i=1}^{m}\bigvee_{p=1}^{k}\left(\alpha'_{i,p} \wedge \beta'_{i,p} \wedge \bigwedge_{j=1}^{k}\omega'_{i,j}\right) \tag{2}$$

$$\vee \bigvee_{i=1}^{m}(\exists x_{k+1})\left(\bigwedge_{p=1}^{k}(x_p \neq x_{k+1}) \wedge \alpha'_i \wedge \beta'_i \wedge \bigwedge_{j=1}^{k+1}\omega'_{i,j}\right). \tag{3}$$

Here $\alpha'_{i,p}$ is false if $\omega'_{i,p} \not\equiv \omega'_{i,k+1}$, otherwise it equals the formula that is obtained from $\alpha'_i$ by replacing each occurence of the variable $x_{k+1}$ by $x_p$.

The subformula (2) is already in the right form. For each disjunct $i$ of (3),

(a) if $\omega'_{i,k+1}$ is $x_{k+1} = \mathbf{c}$ then define $\alpha_i$ as

$$(\exists x_{k+1})(\bigwedge_{p=1}^{k}(x_p \neq x_{k+1}) \wedge x_{k+1} = \mathbf{c} \wedge \alpha'_i),$$

$\beta_i$ as

$$(\exists x_{k+1})(\bigwedge_{p=1}^{k}(x_p \neq x_{k+1}) \wedge x_{k+1} = \mathbf{c} \wedge \beta'_i),$$

and for $j = 1, \ldots, k$, define $\omega_{i,j}$ as $\omega'_{i,j}$.

(b) if $\omega'_{i,k+1}$ is $P_1(x_{k+1})$ then define $\alpha_i$ as

$$(\exists x_{k+1})(\bigwedge_{p=1}^{k}(x_p \neq x_{k+1}) \wedge P_1(x_{k+1}) \wedge \alpha'_i),$$

$\beta_i$ as

$$\begin{aligned}&(\exists x_{k+1})(\bigwedge_{p=1}^{k}(x_p \neq x_{k+1})\\ &\qquad \wedge \neg P_2(x_{k+1}) \wedge x_{k+1} \neq \mathbf{0} \wedge x_{k+1} \neq \mathbf{1} \wedge x_{k+1} \neq \mathbf{2} \wedge \beta'_i),\end{aligned}$$

(the correctness follows from Lemma 6.4(ii)), and for $j = 1, \ldots, k$, define $\omega_{i,j}$ as $\omega'_{i,j}$.

(c) if $\omega'_{i,k+1} = P_2(x_{k+1})$ then define $\alpha_i$ as

$$(\exists x_{k+1})(\bigwedge_{p=1}^{k}(x_p \neq x_{k+1}) \\ \wedge \neg P_1(x_{k+1}) \wedge x_{k+1} \neq \mathbf{0} \wedge x_{k+1} \neq \mathbf{1} \wedge x_{k+1} \neq \mathbf{2} \wedge \alpha'_i),$$

(the correctness follows from Lemma 6.4(i)), $\beta_i$ as

$$(\exists x_{k+1})(\bigwedge_{p=1}^{k} x_p \neq x_{k+1} \wedge P_2(x_{k+1}) \wedge \beta'_i),$$

and for $j = 1, \ldots, k$, define $\omega_{i,j}$ as $\omega'_{i,j}$. ■

In the next lemma we prove that if an FO-formula can only speak about $c_1$ (respectively $c_2$) then this formula in general cannot distinguish between $\xi(\mathcal{N}_1(n_1, n_1))$ and $\xi(\mathcal{N}_1(n_1, n_2))$ (respectively $\xi(\mathcal{N}_2(n_1, n_1))$ and $\xi(\mathcal{N}_2(n_1, n_2))$), where $n_1 \neq n_2$.

**Lemma 6.7** *Let $\xi(x_1, \ldots, x_\ell)$ be a PFP-formula over the vocabulary $\tau_<$. Suppose $\xi$ contains only $m$ distinct variables.*

*(i) Let $\psi$ be an FO-sentence over the vocabulary $\tau_{c,\ell}$ that contains only $m'$ distinct variables and that does not contain the relation symbol $c_2$. Let $n_1 \geq m + m'$. Then for all $n_2, n'_2 \geq m + m'$,*

$$\mathbf{t}(\xi, n_1, n_2) \models \psi \quad \Leftrightarrow \quad \mathbf{t}(\xi, n_1, n'_2) \models \psi.$$

*(ii) Let $\psi$ be an FO-sentence over the vocabulary $\tau_{c,\ell}$ that contains only $m'$ distinct variables and that does not contain the relation symbol $c_1$. Let $n_2 \geq m + m'$. Then for all $n_1, n'_1 \geq m + m'$,*

$$\mathbf{t}(\xi, n_1, n_2) \models \psi \quad \Leftrightarrow \quad \mathbf{t}(\xi, n'_1, n_2) \models \psi.$$

**Proof.** We only prove *(i)*, *(ii)* is symmetric. We can assume w.l.o.g. that $\xi$ and $\psi$ have no variables in common. Let $t(\xi, n_1, n_2)^{\widehat{c_2}}$ be the structure

$t(\xi, n_1, n_2)$ restricted to **0**, **1**, **2** and $c_1$. Since $\psi$ does not speak about $c_2$, it suffices to prove that for all $n_1, n_2, n_2' \geq m + m'$:

$$t(\xi, n_1, n_2)^{\widehat{c_2}} \models \psi \quad \Leftrightarrow \quad t(\xi, n_1, n_2')^{\widehat{c_2}} \models \psi.$$

Suppose there exist $n_2, n_2' \geq m + m'$ such that

$$t(\xi, n_1, n_2)^{\widehat{c_2}} \models \psi \quad \text{and} \quad t(\xi, n_1, n_2')^{\widehat{c_2}} \not\models \psi.$$

Let $\psi'$ be the formula obtained from $\psi$ by replacing each atomic subformula $c_1(z_1, \ldots, z_\ell)$ by $\xi(z_1, \ldots, z_\ell)$. Hence, $\mathcal{N}_1(n_1, n_2) \models \psi'$ and $\mathcal{N}_1(n_1, n_2') \not\models \psi'$. Thus, there exists a sentence with $m + m'$ variables that distinguishes between $\mathcal{N}_1(n_1, n_2)$ and $\mathcal{N}_1(n_1, n_2')$. However, for $n_1, n_2, n_2' \geq m + m'$, using pebble games [19, 7] it is easy to show that $\mathcal{N}_1(n_1, n_2)$ and $\mathcal{N}_1(n_1, n_2')$ are indistinguishable in PFP with $m + m'$ variables. This leads to the desired contradiction. ■

Before starting with the actual proof we show how to simulate several attributes by one attribute.

**Lemma 6.8** *Every synthesized RAG is equivalent to a simple one*

**Proof.** Suppose $\mathcal{R}$ has attributes $a_1, \ldots, a_k$ for the grammar symbol $L$, and attributes *result*, $b_1, \ldots, b_\ell$ for the start symbol $U$. We show that $\mathcal{R}$ is equivalent to the simple RAG $\mathcal{R}'$. To this end, let $r_c = k + 1 + \max\{r_{a_1}, \ldots, r_{a_k}\}$. Assume, w.l.o.g., that none of the variables $y_1, \ldots, y_{r_c}$ occurs in a semantic rule of $\mathcal{R}$. For $i = 1, \ldots, k$, let $\gamma_i(y_1, \ldots, y_{k+1})$ be the formula

$$\gamma_i(y_1, \ldots, y_{k+1}) \ := \ y_i = y_{k+1} \wedge \bigwedge \{y_j \neq y_{k+1} \mid j \in \{1, \ldots, k\} \wedge j \neq i\}.$$

The simulation of the relations $a_1, \ldots, a_k$, by one relation $c$ is the usual one (see, e.g., the proof of the simultaneous induction lemma [7]). We present it in detail as we will refer to it later on. So, for $i = 1, \ldots, k$, let $a_i(0) := \varphi_i$ be the rule in the context $(L \to f, a_i, 0)$. In $\mathcal{R}'$, define $c$ in context $(L \to f, c, 0)$ as

$$c(0) := \{(y_1, \ldots, y_{r_c}) \mid \bigvee_{i=1}^{k} \gamma_i(y_1, \ldots, y_{k+1}) \wedge \varphi_i(y_{k+2}, \ldots, y_{k+1+r_{a_i}})\}.$$

For $i = 1, \dots, k$, let $a_i(0) := \psi_i$ be the rule in context $(L \to L, a_i, 0)$. In $\mathcal{R}'$, define $c$ in context $(L \to L, c, 0)$ as

$$c(0) := \{(y_1, \dots, y_{r_c}) \mid \bigvee_{i=1}^{k} \gamma_i(y_1, \dots, y_{k+1}) \wedge \psi_i'(y_{k+2}, \dots, y_{k+1+r_{a_i}})\},$$

where $\psi_i'$ is obtained from $\psi_i$ by replacing each occurrence of $a_i(1)(\bar{x})$ by

$$(\exists \bar{y})(\exists \bar{z})(\gamma_i(\bar{y}) \wedge c(1)(\bar{y}, \bar{x}, \bar{z})),$$

where $\bar{z}$ and $\bar{x}$ have no variables in common.

Finally, let *result*$(0) := \sigma$ be the rule in context $(U \to LL, \mathit{result}, 0)$, and let for $i = 1, \dots, \ell$, $b_i(0) := \sigma_i$ be the rule in context $(U \to LL, b_i, 0)$. Assume, w.l.o.g., that $\sigma$, $\sigma_1$, …, $\sigma_\ell$ have no variables in common. Let $\sigma'$ be the formula obtained from $\sigma$ by replacing each occurrence of $b_i(0)(\bar{y})$ by $\sigma_i(\bar{y})$. Then define *result* in $\mathcal{R}'$ by the formula obtained from $\sigma'$ by replacing each occurrence of $a_i(j)(\bar{x})$ by $(\exists \bar{y})(\exists \bar{z})(\gamma_i(\bar{y}) \wedge c(j)(\bar{y}, \bar{x}, \bar{z}))$, where $\bar{z}$ and $\bar{x}$ have no variables in common.

It follows that for any tree $\mathbf{t}$ with root $\mathbf{r}$, $\mathcal{R}(\mathbf{t})(\mathit{result}(\mathbf{r}))$ is true if and only if $\mathcal{R}'(\mathbf{t})(\mathit{result}(\mathbf{r}))$ is true. ■

By putting all the pieces together we can prove Theorem 4.9. Indeed, towards a contradiction suppose `equal_subtree` is expressible by a synthesized RAG. Suppose $\mathcal{R}$ has attributes $a_1$, …, $a_k$ for the grammar symbol $L$, and attributes *result*, $b_1$, …, $b_\ell$ for the start symbol $U$. W.l.o.g., we can assume that $a_1$ is a set-valued attribute that contains for each node all its descendants: for production $L \to \ell$ define $a_1(0) := \{\mathbf{0}, \mathbf{1}\}$, and for production $L \to L$ define $a_1(0) := \{\mathbf{0}\} \cup a_1(1)$.

By Lemma 6.8, $\mathcal{R}$ is equivalent to the simple RAG $\mathcal{R}'$. It, hence, suffices to show that $\mathcal{R}'$ cannot express `equal_subtree`.

According to Lemma 6.3, there exists a PFP-formula $\xi(x_1, \dots, x_{r_c})$ such that for all $n_1, n_2 > 1$

$$\mathcal{R}'(\mathbf{t}(n_1, n_2))(c(\mathbf{n}_1)) = \xi(\mathcal{N}_1(n_1, n_2)),$$

and

$$\mathcal{R}'(\mathbf{t}(n_1, n_2))(c(\mathbf{n}_2)) = \xi(\mathcal{N}_2(n_1, n_2)),$$

where $\mathbf{n}_1$ is the first child and $\mathbf{n}_2$ is the second child of the root. Let $\psi$ be the sentence that defines *result* in $\mathcal{R}'$. Then for all $n_1, n_2 > 1$,

$$\mathcal{R}'(\mathbf{t}(n_1, n_2))(\mathit{result}(\mathbf{r})) \text{ is true} \quad \Leftrightarrow \quad \mathbf{t}(\xi, n_1, n_2) \models \psi', \tag{4}$$

where $\psi'$ is obtained from $\psi$ by replacing each occurrence of $c(1)(\bar{x})$ by $c_1(\bar{x})$ and each occurrence of $c(2)(\bar{x})$ by $c_2(\bar{x})$, and where $\mathbf{r}$ is the root of $\mathbf{t}(\xi, n_1, n_2)$. Now, define $P_1(x)$ as

$$P_1(x) := (\exists \bar{y})(\exists \bar{z})(\gamma_1(\bar{y}) \wedge c_1(\bar{y}, x, \bar{z}) \wedge x \neq \mathbf{0} \wedge x \neq \mathbf{1} \wedge x \neq \mathbf{2}),$$

and $P_2(x)$ as

$$P_2(x) := (\exists \bar{y})(\exists \bar{z})(\gamma_1(\bar{y}) \wedge c_2(\bar{y}, x, \bar{z}) \wedge x \neq \mathbf{0} \wedge x \neq \mathbf{1} \wedge x \neq \mathbf{2}).$$

Note that $P_1$ and $P_2$ just define the value of $a_1(\mathbf{n}_1)$ and $a_1(\mathbf{n}_2)$, respectively. Here, $\gamma_1$ is the formula defined in the proof of Lemma 6.8 to encode the attributes $a_1, \ldots, a_k$ into $c$.

Then for all $n_1, n_2 > 1$,

$$\mathbf{t}(\xi, n_1, n_2) \models P_1[\mathbf{n}] \quad \Leftrightarrow \quad \mathbf{n} \in \{1, \ldots, n_1 - 1\},$$

and

$$\mathbf{t}(\xi, n_1, n_2) \models P_2[\mathbf{n}] \quad \Leftrightarrow \quad \mathbf{n} \in \{n_1 + 1, \ldots, n_1 + n_2 - 1\}.$$

Hence, by Lemma 6.6, $\psi'$ is equivalent to a sentence of the form $\bigvee_{i=1}^{n} \alpha_i \wedge \beta_i$, where the $\alpha_i$'s are sentences that do not contain $c_2$, and the $\beta_i$'s are sentences that do not contain $c_1$ (since there are no free variables, there are no $\omega$'s). W.l.o.g., we can assume that $\xi$ and $\bigvee_{i=1}^{n} \alpha_i \wedge \beta_i$ have no variables in common. Let $m$ be the number of variables in $\xi$, and let $m'$ be the number of variables in $\bigvee_{i=1}^{n} \alpha_i \wedge \beta_i$. By a simple counting argument, there have to exist $n_1' > n_1 \geq m + m'$ such that for all $i = 1, \ldots, n$:

$$\mathbf{t}(\xi, n_1, m + m') \models \alpha_i \quad \Leftrightarrow \quad \mathbf{t}(\xi, n_1', m + m') \models \alpha_i.$$

Hence, by applying Lemma 6.7(i) twice, for $i = 1, \ldots, n$:

$$\mathbf{t}(\xi, n_1, n_1) \models \alpha_i \quad \Leftrightarrow \quad \mathbf{t}(\xi, n_1', n_1) \models \alpha_i.$$

From Lemma 6.7(ii), it follows that for $i = 1, \ldots, n$:

$$\mathbf{t}(\xi, n_1, n_1) \models \beta_i \quad \Leftrightarrow \quad \mathbf{t}(\xi, n_1', n_1) \models \beta_i.$$

Hence,

$$\mathbf{t}(\xi, n_1, n_1) \models \bigvee_{i=1}^{n} \alpha_i \wedge \beta_i \quad \Leftrightarrow \quad \mathbf{t}(\xi, n_1', n_1) \models \bigvee_{i=1}^{n} \alpha_i \wedge \beta_i.$$

But then by (4), we have that

$$\mathcal{R}'(\mathtt{t}(n_1, n_1))(\mathit{result}(\mathbf{r})) \text{ is true} \quad \Leftrightarrow \quad \mathcal{R}'(\mathtt{t}(n_1, n_1'))(\mathit{result}(\mathbf{r})) \text{ is true},$$

and $n_1 \neq n_1'$. Hence, $\mathcal{R}'$ does not express `equal_subtree`. This concludes the proof of Theorem 4.9.

# Appendix B

In this appendix we show that the query $\mathcal{Q}$ in the proof of Theorem 4.14 is expressible by a synthesized RAG.

The RAG computing $\mathcal{Q}$ uses the following synthesized attributes for all non-terminals:

1. $X$ is a Boolean attribute: $X(\mathbf{n})$ is true if there is a node labeled $X$ among the descendants of $\mathbf{n}$ (note that $X$ is a non-terminal);

2. *chain* is a Boolean attribute:

   (a) $\mathit{chain}(\mathbf{n})$ is false if there is no $X$-labeled node among the descendants of $\mathbf{n}$, or if there is no chain of occurrences starting on the the first $X$-labeled node in the preorder traversal of the subtree with root $\mathbf{n}$;

   (b) $\mathit{chain}(\mathbf{n})$ is true if the length of the chain of occurrences of $(\bar{p}, \bar{k})$ starting at the first $X$-labeled node in the preorder traversal of the subtree with root $\mathbf{n}$ is a power of two. Note that if this node is not derived by $p_1$, then the length of the chain of occurrences starting at that node is 1, which is a power of two.

3. $D$ is a set-valued attribute: $D(\mathbf{n})$ contains $\mathbf{n}$ and all descendants of $\mathbf{n}$.

4. $<$ is a binary attribute: $<(\mathbf{n})$ is a total order on $D(\mathbf{n})$.

5. *occ* is a Boolean attribute: *occ* is true if $\mathbf{n}$ is derived with $p_i$, for some $i \in \{1, \ldots, m\}$, and there exist nodes $\mathbf{n}_{i+1}, \ldots, \mathbf{n}_{m+1}$, such that a chain of occurrences starts at $\mathbf{n}_{m+1}$, and for $j = i+1, \ldots, m$, $\mathbf{n}_j$ is derived with $p_j$, and $\mathbf{n}_{j+1}$ is the $k_j$-th child of $\mathbf{n}_j$.

6. *is*-$p_1$ is a Boolean attribute: *is*-$p_1(\mathbf{n})$ is true if $\mathbf{n}$ is derived with $p_1$.

7. $a$ is a set-valued attribute: $a(\mathbf{n})$ is a subset of $D(\mathbf{n})$.

   (a) If *occ* is false, then $a(\mathbf{n})$ is empty;

   (b) If *occ* is true and $\mathbf{n}$ is not derived by $p_1$, then the nodes in $a(\mathbf{n})$ encode, w.r.t. $<(\mathbf{n})$, in binary the length of the chain of occurrences of $(\bar{p}, \bar{k})$ starting at $\mathbf{n}_{m+1}$, where $\mathbf{n}_{m+1}$ is as defined in 5: if $a(\mathbf{n})$ contains the nodes $\mathbf{n}_1, \ldots, \mathbf{n}_r$, and these occur respectively in the

$i_1$-th, ..., $i_k$-th position in the ordering $<(\mathbf{n})$, then $a(\mathbf{n})$ encodes the number $\Sigma_{p=1}^{r} 2^{i_p - 1}$.

(c) If *occ* is true and $\mathbf{n}$ is derived by $p_1$, then the nodes in $a(\mathbf{n})$ encode, w.r.t. $<(\mathbf{n})$, in binary the length of the chain of occurrences of $(\bar{p}, \bar{k})$ starting at $\mathbf{n}$.

The RAG is now defined as follows:

1. Consider the production $p = X_0 \to X_1 \dots X_n$ not in $\bar{p}$. Define $T(p) = \{i \in \{1, \dots, n\} \mid X_i \text{ is a terminal}\}$. We write $i \notin T(p)$ as a shorthand for $i \in \{1, \dots, n\} - T(p)$. Define

$$X(0) \;:=\; \begin{cases} \text{true} & \text{if } X_0 = X; \\ \bigvee_{i \notin T(p)} X(i) & \text{otherwise;} \end{cases}$$

and

$$\mathit{chain}(0) \;:=\; \begin{cases} \text{true} & \text{if } X_0 = X; \\ \gamma & \text{otherwise,} \end{cases}$$

where $\gamma$ is an FO-sentence whose truth value equals that of $\mathit{chain}(i)$, where $i$ is the smallest such that $X(i)$ is true; if such an $i$ does not exists, $\gamma$ is false. For $i = 1, \dots, n$, let $E(i)$ be $D(i)$ if $i \notin T(p)$, and $\{(\mathbf{i})\}$ when $i \in T(p)$. Then define

$$D(0) \;:=\; \bigcup_{i=1}^{n} E(i) \cup \{(\mathbf{0})\};$$

$$\begin{aligned} <(0) \;:=\; & \bigcup_{i \notin T(p)} <(i) \cup (D(0) \times \{(\mathbf{0})\}) \cup \bigcup_{i \in T(p)} \{(\mathbf{i}, \mathbf{i})\} \\ & \cup \bigcup \{E(i) \times E(j) \mid i, j \in \{1, \dots, n\} \wedge i < j\}; \end{aligned}$$

$$\mathit{occ}(0) \;:=\; \text{false};$$
$$\mathit{is\text{-}p_1}(0) \;:=\; \text{false};$$

and

$$a(0) \;:=\; \emptyset.$$

If $X_0 = U$ then define

$$\mathit{result}(0) \;:=\; X(0) \to \mathit{chain}(0).$$

2. For $p_1 = X_0 \to X_1 \ldots X_{n_1}$, define

$$X(0) \;:=\; \text{true};$$

and

$$\mathit{chain}(0) \;:=\; \mathit{occ}(0) \wedge (\exists x)\Big(a(0)(x) \wedge (\forall y)(a(0)(y) \to x = y)\Big).$$

The attribute *chain* becomes true if $\mathit{occ}(0)$ is true and if $a(0)$ contains exactly one element, i.e., $a(0)$ encodes a number which is a power of two. Define,

$$D(0) \;:=\; \bigcup_{i=1}^{n_1} E(i) \cup \{(\mathbf{0})\};$$

$$\begin{aligned} <(0) \;:=\; & \bigcup_{i \notin T(p_1)} <(i) \cup \bigcup_{i \in \{1,\ldots,n_1\}-\{k_1\}} (D(k_1) \times E(i)) \\ & \cup (D(0) \times \{(\mathbf{0})\}) \cup \bigcup_{i \in T(p_1)} \{(\mathbf{i}, \mathbf{i})\} \\ & \cup \bigcup \{E(i) \times E(i') \mid \\ & \qquad i, i' \in \{1, \ldots, n_1\},\ i < i',\ i \neq k_1 \wedge i' \neq k_1\}; \end{aligned}$$

The reason for this definition is that, in order to correctly represent the number in $a(k_j)$, we have to make sure that all elements not in $D(k_j)$ come after the elements in $D(k_j)$ in the ordering $<(0)$. Finally define

$$\begin{aligned} \mathit{occ}(0) \;&:=\; \mathit{occ}(k_1); \\ \mathit{is}\text{-}p_1(0) \;&:=\; \text{true}; \end{aligned}$$

and

$$a(0) \;:=\; \mathit{occ}(0) \wedge \varphi(z, a(k_1), <(0)),$$

where $\varphi(z, Z, <)$ is the formula

$$\begin{aligned} & (\exists z')\Big(Z(z') \wedge \neg Z(\mathrm{Succ}(z')) \wedge (\forall z'')(z'' < z' \to Z(z'')) \\ & \qquad\qquad \wedge \big(z = \mathrm{Succ}(z') \vee (z' < z \wedge Z(z))\big)\Big) \\ & \vee (\exists z')\Big(\mathrm{First}(z') \wedge \neg Z(z') \wedge (z = z' \vee Z(z))\Big), \end{aligned}$$

with Succ the successor function, and First is the first element in the ordering $<$. This formula augments the number in $a(k_1)$ with 1.

3. For $j = 2, \ldots, m-1$, define for $p_j = X_0 \to X_1 \ldots X_{n_j}$,

   $$X(0) \;:=\; \bigvee_{i \notin T(p_j)} X(i)$$

   and

   $$\mathit{chain}(0) \;:=\; \gamma,$$

   where $\gamma$ is defined as in (1). Further, define

   $$D(0) \;:=\; \bigcup_{i=1}^{n_j} E(i) \cup \{(\mathbf{0})\};$$

   $$\begin{aligned} <(0) \;:=\; & \bigcup_{i \notin T(p_j)} <(i) \cup \bigcup_{i \in \{1,\ldots,n_j\} - \{k_j\}} (D(k_j) \times E(i)) \\ & \cup (D(0) \times \{(\mathbf{0})\}) \cup \bigcup_{i \in T(p_j)} \{(\mathbf{i}, \mathbf{i})\} \\ & \cup \bigcup \{E(i) \times E(i') \mid \\ & \qquad i, i' \in \{1, \ldots, n_j\},\ i < i',\ i \neq k_j \wedge i' \neq k_j\}; \end{aligned}$$

   $$\begin{aligned} \mathit{occ}(0) \;&:=\; \mathit{occ}(k_j); \\ \mathit{is}\text{-}p_1(0) \;&:=\; \text{false}; \end{aligned}$$

   and

   $$a(0) \;:=\; \mathit{occ}(0) \wedge a(k_j)(z).$$

4. For $p_m = X_0 \to X_1 \ldots X_{n_m}$, define

   $$X(0) \;:=\; \text{true};$$

   and

   $$\mathit{chain}(0) \;:=\; \gamma.$$

   Here $\gamma$ is defined as in (1). Define,

   $$D(0) \;:=\; \bigcup_{i=1}^{n_m} E(i) \cup \{(\mathbf{0})\};$$

$$
\begin{aligned}
<\!(0) \;&:=\; \bigcup_{i \notin T(p_1)} <\!(i) \cup \bigcup_{i \in \{1,\ldots,n_m\}-\{k_m\}} (D(k_m) \times E(i)) \\
&\cup (D(0) \times \{(\mathbf{0})\}) \cup \bigcup_{i \in T(p_1)} \{(\mathbf{i},\mathbf{i})\} \\
&\cup \bigcup \{E(i) \times E(i') \mid \\
&\qquad i, i' \in \{1,\ldots,n_m\},\; i < i',\; i \neq k_m \wedge i' \neq k_m\};
\end{aligned}
$$

$$
\begin{aligned}
\mathit{occ}(0) \;&:=\; \neg \mathit{is}\text{-}p_1(k_m) \vee \mathit{occ}(k_m); \\
\mathit{is}\text{-}p_1(0) \;&:=\; \text{false};
\end{aligned}
$$

and define

$$a(0) \;:=\; \sigma.$$

Here, $\sigma$ is an FO-sentence that expresses the following: $a(0)$ contains the singleton consisting of the first element in $<\!(0)$ if $\mathit{is}\text{-}p_1(k_m)$ is false; $a(0)$ equals $a(k_m)$ if $\mathit{occ}(k_m)$ is true; and $a(0) = \emptyset$ otherwise.