On Implementing Structured Document Query Facilities on Top of a DOOD

Frank Neven* and Jan Van den Bussche

University of Limburg, Departement WNI,
Universitaire Campus, B-3590 Diepenbeek, Belgium.
email: {fneven,vdbuss}@luc.ac.be

Abstract. The paper consists of two parts. In the first part we introduce a model for structured document databases where we propose Boolean-valued attribute grammars (BAGs) as a query facility. In the second part we show that DOOD technology offers a natural platform on top of which this model as a whole can be conveniently implemented. Each structured document database is mapped to an OO-database and each BAG is translated into deductive rules. Our translation is such that the well-founded semantics and the naive bottom-up fixpoint procedure of the deductive rules capture the evaluation of the BAG. We also present a modification of the translation suited to the inflationary semantics.

1 Introduction

As originally proposed by Gonnet and Tompa [8], a structured document database can be naturally modeled as a derivation tree of some context-free grammar, which functions as the "schema" of the database. Every terminal symbol of the grammar has an associated data type (e.g., `long string`, `number`, `image`), and every leaf node of the tree holds a data value (e.g., an ASCII text, a picture) of the type of the terminal symbol that labels the node; this value is no longer part of the main structure of the document and is considered to be atomic as far as the database is concerned.[1]

Under this view of document databases, a practical way of thinking about a *query* is as 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 by the query. In this paper, we propose the use of *Boolean-valued attribute grammars (BAGs)* for expressing such queries. Attribute grammars, introduced by Knuth [10], provide a well-established mechanism for annotating the nodes of a derivation tree with so-called "attribute values", 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). BAGs are a specialization of attribute grammars to Boolean-valued attribute values only, and

---

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

[1] Of course, to specific helper programs, such as image processing software, the value is not atomic.

to semantic rules in the form of propositional logic formulas. 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.

Our goal in this paper is to introduce the BAG model and to propose a design to implement the structured document query facility provided by BAGs. More specifically, we want to show that DOOD technology offers a natural platform on top of which such an implementation becomes remarkably straightforward.

Concretely, we represent a context-free grammar by an OODB schema, so that structured documents (i.e., derivation trees of the grammar augmented with atomic values stored in the leaf nodes) can be stored as instances over this OODB schema. We then translate a BAG into a set of deductive rules, which in general contain negation. Our main technical result states that the *well-founded sematics* [14, 2] of this set of rules equals exactly the semantics of the BAG. More precisely, the value for an attribute a of a node $\mathbf{n}$ in a tree $\mathbf{T}$, as derived by the BAG, will be true, if and only if the fact $a(\mathbf{n})$ is in the (total) well-founded model of the deductive rules applied to the database $\mathbf{T}$.

Second, we prove the somewhat surprising result that in the above, we can as well use the naive bottom-up fixpoint procedure instead of the well-founded semantics. This procedure (the standard one for deductive programs without negation) is usually not considered in the presence of negation, as it is not even guaranteed to terminate; nevertheless, we show that it does on the programs generated by our translation and that it also captures the semantics of the BAG.

Finally, we also consider the inflationary fixpoint semantics [3, 11, 2]. This semantics is simpler than the well-founded semantics and more common in research on the expressiveness of query languages. An advantage of the inflationary semantics is that once a fact is derived it never needs to be retracted. It is thus interesting to point out that there is also a translation of BAGs to deductive rules whose inflationary fixpoint equals exactly the semantics of the BAG. In fact, the well-founded semantics and the naive bottom-up fixpoint evaluation also coincide with the semantics of the BAG on this translation.

This paper is further organized as follows. In Sections 2 and 3, we introduce our model of structured document databases and of queries to these. In Section 4, we discuss the OODB representation of structured documents. In Section 5, we present the different translations of BAGs into deductive rule programs and examine their properties. We conclude the paper in Section 6 with a discussion of some related work.

2 Structured Document Databases

Consider a context-free grammar $G = (N, T, P, S)$, where N is the set of non-terminal symbols, T is the set of terminal symbols, P is the set of productions, and S is the start symbol. Assume that to each terminal $X \in T$ a basic data type $\mathit{type}(X)$ is associated; $\mathit{type}(X)$ can be `long string`, or `number`, or `image`, or any other basic data type supported by the computer system.

Then G, together with the type assignment *type*(), serves as a *schema* for structured document databases. An *instance* of this schema then is a derivation tree of G, where each leaf node $\mathbf{n}$ (labeled by some terminal X) additionally holds a data value $val(\mathbf{n})$ of type $type(X)$. Thus, the internal nodes (labeled by non-terminals) identify the structural elements of the document database, while the terminal nodes contain the actual contents.

Example 1. As an example, consider Fig. 1. The figure shows a schema modeling a list of researcher's Web pages. Each page contains the name of the researcher, a section containing some personal information, and a section containing a list of publications. Terminal symbols are underlined. An example instance of this schema is shown in Fig. 2. □

$$
\begin{array}{lll}
\text{HPs} & \longrightarrow & \text{HP} \quad \text{HPs} \\
\text{HPs} & \longrightarrow & \varepsilon \\
\text{HP} & \longrightarrow & \underline{\text{Name}} \quad \text{Personal} \quad \text{Publs} \\
\text{Personal} & \longrightarrow & \underline{\text{Picture}} \quad \underline{\text{Position}} \\
\text{Publs} & \longrightarrow & \text{Pub} \quad \text{Publs} \\
\text{Publs} & \longrightarrow & \varepsilon \\
\text{Pub} & \longrightarrow & \underline{\text{Title}} \quad \text{Authors} \quad \underline{\text{Journal}} \quad \underline{\text{Year}} \\
\text{Authors} & \longrightarrow & \underline{\text{Name}} \\
\text{Authors} & \longrightarrow & \underline{\text{Name}} \quad \text{Authors.}
\end{array}
$$

$$
\begin{array}{ll}
type(\text{Name}) = \texttt{string}; & type(\text{Title}) = \texttt{string}; \\
type(\text{Picture}) = \texttt{image}; & type(\text{Journal}) = \texttt{string}; \\
type(\text{Position}) = \texttt{string}; & type(\text{Year}) = \texttt{number}.
\end{array}
$$

Fig. 1. A schema modeling a collection of Web pages. (The symbol ε denotes an empty production.)

The reader may have noted that, in principle, the type assignment *type*() in a schema and the value assignment *val*() in an instance can always be avoided. Indeed, in the end every data value is a string of bits, which could be directly incorporated in the main database structure if the context-free grammar contains productions

$$
\begin{array}{lll}
\text{Bitstring} & \longrightarrow & \text{Bit} \quad \text{Bitstring} \\
\text{Bitstring} & \longrightarrow & \varepsilon \\
\text{Bit} & \longrightarrow & 0 \\
\text{Bit} & \longrightarrow & 1
\end{array}
$$

However, it is of course not practical to incorporate all data in the document, until the last bit, into its main structure; in general, the choice of which data to incorporate and which data to model using *val*() is a matter of schema design and will depend on the intended application.

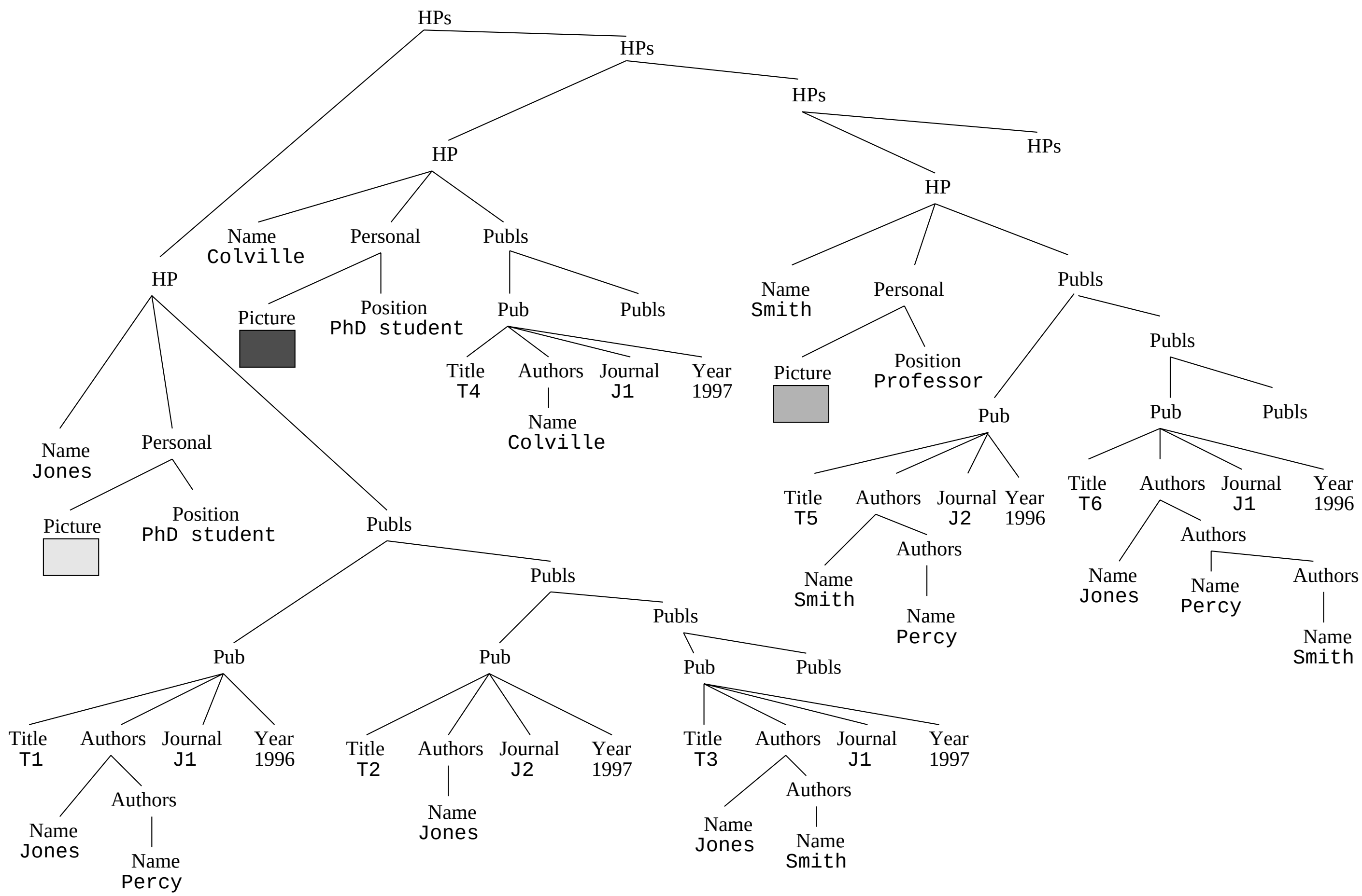

Fig. 2. An instance over the schema in Fig. 1.

3 Structured Document Queries

Conceptually, we define a *query* Q over some schema G as a mapping on the instances $\mathbf{T}$ of G, such that $Q(\mathbf{T})$ is a set of nodes of the tree $\mathbf{T}$. For example, using our example schema, a query could ask for the homepage and the first publication of each PhD student having at least two publications. The answer of this query is thus a set of HP-labeled nodes and Pub-labeled nodes.

We will use Boolean-valued attribute grammars (BAGs) to express such queries. A BAG $\mathcal{B}$ over G consists of

- a set A of symbols called *attributes*;
- mappings Syn and Inh from $N \cup T$ (the non-terminal and terminal symbols of G) to subsets of A; and
- a set R of *semantic rules*.

For any symbol X of G, $\mathrm{Syn}(X)$ is called the set of *synthesized attributes of* X and $\mathrm{Inh}(X)$ is called the set of *inherited attributes of* X.[2]

On any given instance $\mathbf{T}$ of G, and for any node $\mathbf{n}$ of $\mathbf{T}$, if $\mathbf{n}$ is labeled X, the BAG will derive a Boolean value $a(\mathbf{n})$ for each attribute a (synthesized or inherited) of X. This is done by the semantic rules; we now explain their syntax and semantics.

Let $p : X_0 \to X_1 \ldots X_n$ be a production of G, let a be an attribute, and let $i \in \{0, \ldots, n\}$. The triple (p, a, i) is called a *context* if

- $i = 0$ and a is a synthesized attribute of X_0, or
- $i > 0$ and a is an inherited attribute of X_i.

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

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

where φ is a propositional logic formula over the propositions $b(j)$, with $j \in \{0, \ldots, n\}$ and b an attribute (synthesized or inherited) of X_j, as well as any built-in predicates on atomic data values. For example, if X_j is a terminal symbol with $\mathit{type}(X_j) = \mathtt{image}$, then $\mathtt{width}(j) < 490$ could occur in φ as a built-in predicate on image data values.

The set R of semantic rules of a BAG must contain exactly one rule for each context (p, a, i) as above. Inspecting the definition of context, one notes that if a is a synthesized attribute, the semantic rule defines the value of a for a node in terms of previously defined attribute values of the node itself and of its children; hence, synthesized attributes are derived bottom-up. If a is inherited, the rule defines the value in terms of attributes of the node itself, its siblings, and its parent; hence, inherited attributes are derived top-down.

[2] Technically, we require that $\mathrm{Syn}(X)$ and $\mathrm{Inh}(X)$ are disjoint for each X, that the start symbol has no inherited attributes, and that terminal symbols do not have synthesized attributes.

Example 2. Before we formally describe this derivation process, we already give an example of a BAG over our example schema in Fig. 3. This BAG expresses the example query mentioned earlier: "give all homepages and the first publication of each PhD student having at least two publications". One of the attributes defined by the BAG is the special attribute **select**; the nodes for which the value of this attribute is true form the answer set of the query. The other elements of the set A of attributes defined by this BAG are *PhD*, *atleast2*, and *selected*. We have $\mathrm{Syn}(\mathrm{HP}) = \{\mathbf{select}\}$; $\mathrm{Syn}(\mathrm{Personal}) = \{\mathit{PhD}\}$; $\mathrm{Syn}(\mathrm{Publs}) = \{\mathit{atleast2}, \mathit{atleast1}\}$; $\mathrm{Inh}(\mathrm{Publs}) = \{\mathit{selected}\}$, and $\mathrm{Inh}(\mathrm{Pub}) = \{\mathbf{select}\}$; all other sets $\mathrm{Syn}(X)$ or $\mathrm{Inh}(X)$ are empty.

To see how this BAG indeed expresses the above query on any given instance, we start by looking at the rules defining the result attribute **select**.

The rule $\mathbf{select}(0) := \mathit{PhD}(2) \wedge \mathit{atleast2}(3)$ for the production of HP nodes expresses that a homepage will be selected if the attribute *PhD* is true for its Personal section and the attribute *atleast2* is true for its Publs section. Note that in semantic rules, nodes are numbered; 0 stands for the parent node of the production, and i stands for its ith child node.

The rule $\mathit{PhD}(0) := (2 = \texttt{'PhD student'})$ for the production of Personal nodes expresses that *PhD* is true for a Personal node if the data value of its Position child node (of type `string`, cf. Figure 1) equals `PhD student`. Here, the comparison $(2 = \texttt{'PhD student'})$ is a built-in predicate on `string` data values (recall from the definition that such built-in predicates can be used in the defining formula of a semantic rule).

The rule $\mathit{atleast2}(0) := \mathit{atleast1}(2)$ for the production Publs $\rightarrow$ Pub Publs expresses that a list of publications has at least two elements (attribute *atleast2*) if its tail has at least one (attribute *atleast1*). The two rules for *atleast1* set this attribute to true in all Publs nodes except for those representing an empty list (generated by the production Publs $\rightarrow \varepsilon$).

The query also selects for each selected homepage the first publication in its publication list. This is achieved by the inherited attributes **select** of Pub and *selected* of Publs (note that the special attribute **select** is synthesized for HP, but inherited for Pub). The rule $\mathbf{select}(1) := \mathit{selected}(0)$ for the production Publs $\rightarrow$ Pub Publs expresses that a publication node is selected if the attribute *selected* of its parent node is true. By the rules defining *selected*, the latter holds precisely if that parent node is the publication list of a selected homepage, so that indeed the first publication of that selected homepage will be selected as well. □

We finish this section with a formal description of the evaluation of a BAG on an instance $\mathbf{T}$. We must define truth values for all attributes $a(\mathbf{n})$, where $\mathbf{n}$ is a node of $\mathbf{T}$ and a is an attribute of the grammar symbol that labels $\mathbf{n}$. Initially, all attribute values are undefined. We then repeatedly choose a node $\mathbf{n}_0$ labeled X_0, with children $\mathbf{n}_1, \ldots, \mathbf{n}_n$ labeled $X_1, \ldots, X_n$ respectively. Let $p : X_0 \rightarrow X_1 \ldots X_n$ be the corresponding production of G. Choose a rule $a(i) := \varphi$ in some context (p, a, i), such that for each proposition $b(j)$ occurring in φ, the truth value of $b(\mathbf{n}_j)$ has already been determined. Possible built-in

$$
\begin{array}{ll}
\mathrm{HP} \to \underline{\mathrm{Name}}\ \mathrm{Personal}\ \mathrm{Publs} & \mathbf{select}(0) := \mathit{PhD}(2) \wedge \mathit{atleast2}(3) \\
 & \mathit{selected}(3) := \mathbf{select}(0) \\
\mathrm{Personal} \to \underline{\mathrm{Picture}}\ \underline{\mathrm{Position}} & \mathit{PhD}(0) := (2 = \texttt{'PhD student'}) \\
\mathrm{Publs} \to \mathrm{Pub}\ \mathrm{Publs} & \mathit{atleast2}(0) := \mathit{atleast1}(2) \\
 & \mathit{atleast1}(0) := \mathrm{true} \\
 & \mathbf{select}(1) := \mathit{selected}(0) \\
 & \mathit{selected}(2) := \mathrm{false} \\
\mathrm{Publs} \to \varepsilon & \mathit{atleast1}(0) := \mathrm{false} \\
 & \mathit{atleast2}(0) := \mathrm{false}
\end{array}
$$

Fig. 3. Select all homepages and the first publication of PhD students having at least two publications.

predicates $f(j_1, \ldots, j_k)$ occurring in φ, with $X_{j_1}, \ldots, X_{j_k}$ terminal symbols, are interpreted as $f(\mathit{val}(\mathbf{n}_{j_1}), \ldots, \mathit{val}(\mathbf{n}_{j_k}))$. Thus we can evaluate the complete formula φ and define the truth value of $a(\mathbf{n}_i)$ as the result of this evaluation.

If the above evaluation process completely defines all attribute values on each possible instance $\mathbf{T}$, this means that the attribute definitions as given by the semantic rules are sound. This soundness property of BAGs is known as *non-circularity.* (Non-circularity is well known to be effectively decidable [10].) Obviously we are not interested in unsound BAGs; all BAGs will be implicitly assumed to be non-circular in the sequel.

Example 3. Figure 4 shows a fragment of the result of the BAG of Fig. 3 evaluated on the instance of Example 1. □

4 Structured Document Databases as OODB Instances

In this section we present a way in which structured document database instances can be stored in OODBs. The OO modeling features we assume are supported by any standard OODB system; for concreteness we will use the terminology of the O_2 data model [9] in what follows.

Let $\mathbf{T}$ be an instance of schema G, and let $\mathbf{n}_0$ be a node in $\mathbf{T}$.

- If $\mathbf{n}_0$ is labeled by a terminal symbol X of G, $\mathbf{n}_0$ is a leaf node and can be viewed as representing a basic value $\mathit{val}(\mathbf{n}_0)$ of type $\mathit{type}(X)$.
- If $\mathbf{n}_0$ is labeled by a non-terminal symbol, then it was generated by some production $X_0 \to X_1 \ldots X_n$ of G and has n children $\mathbf{n}_1, \ldots, \mathbf{n}_n$. In this case $\mathbf{n}_0$ can be viewed as the identifier of an object whose value is the tuple $[\mathbf{n}_1, \ldots, \mathbf{n}_n]$.

Under this OODB representation of structured document instances, what is the underlying OODB representation of the structured document schema G?

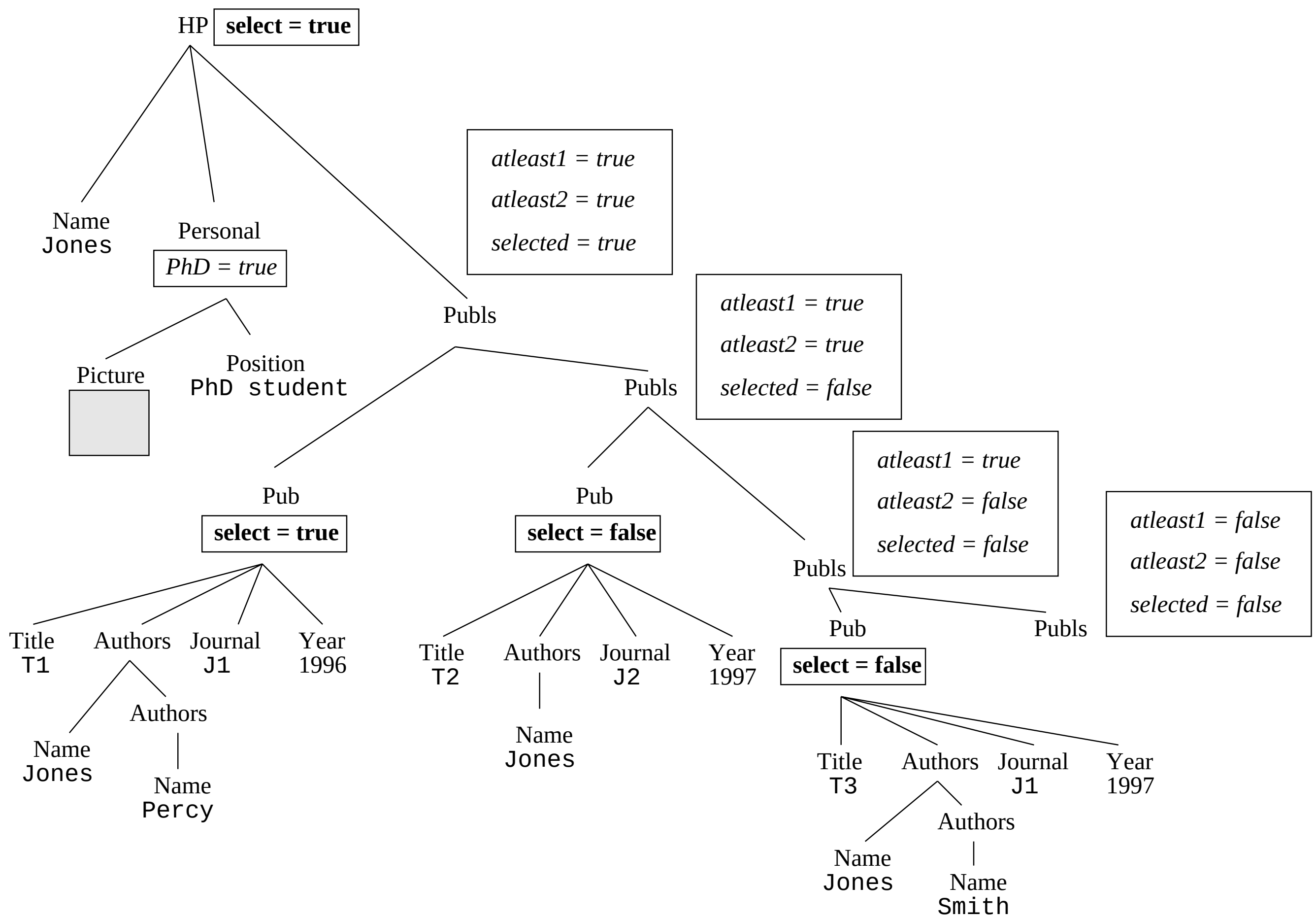

Fig. 4. Fragment of the result of the BAG of Fig. 3.

The most natural would be simply to have a distinct class for each non-terminal symbol X of G, such that in an instance all nodes labeled X would be the objects of that class. However, such nodes can be generated by different productions of G and thus can have completely different tuple values. Hence, we must create, for each production $X \to \ldots$ of G generating X-labeled nodes, a subclass of the class for X in which the nodes generated by this production are contained. The original class for X then serves as a common superclass for all nodes labeled X. Note that this class is "abstract" (in the sense of, e.g., Smalltalk [7]) in that it contains no direct elements; all objects of class X really belong to some subclass (production) of X.

Formally, the OODB schema representing a structured document schema G has as class names all non-terminals of G and all productions of G. The class hierarchy is such that a production of the form $X \to \ldots$ is a subclass of X. The type of the classes are as follows: for each production $p : X_0 \to X_1 \ldots X_n$, the type of class p is the tuple type

$$[\tilde{X}_1, \ldots, \tilde{X}_n],$$

where $\tilde{X}_i$ denotes simply X_i itself if X_i is non-terminal, and $\tilde{X}_i$ denotes the basic type $\textit{type}(X_i)$ if X_i is terminal.[3]

Example 4. In the OODB representation of our example schema, we have an abstract superclass 'Authors' for the non-terminal Authors, with subclasses 'Authors $\to$ Name Authors' of type [`string`, Authors] and 'Authors $\to$ Name' of type [`string`]. □

5 Translating Structured Document Queries into Deductive Rule Programs

Given a query (expressed as a BAG) on a structured document (stored as an OODB instance), we want to compute the answer of the query preferably using the query language of the OODB system. However, a first observation to be made is that a query language with usual, first-order, expressive power (such as SQL and its derivatives) does not suffice for this purpose. This is illustrated in the following example.

Example 5. Over our example schema, consider the following BAG expressing the query "give all co-authors of Jones":

HP $\to$ Name Personal Publs	$\textit{Jones}(3) := (1 = \texttt{'Jones'})$
Publs $\to$ Pub Publs	$\textit{Jones}(1) := \textit{Jones}(0)$;
	$\textit{Jones}(2) := \textit{Jones}(0)$
Pub $\to$ Title Authors Journal Year	$\textit{Jones}(2) := \textit{Jones}(0)$
Authors $\to$ Name	$\textbf{select}(1) := \textit{Jones}(0)$
Authors $\to$ Name Authors	$\textbf{select}(1) := \textit{Jones}(0)$;
	$\textit{Jones}(2) := \textit{Jones}(0)$

[3] The type of the abstract superclasses X, with X a non-terminal, is unimportant for the purpose of the present discussion.

This query is not expressible in SQL.[4] □

However, the iterative evaluation process of a BAG, described at the end of Section 3, can be computed using deductive rules, as we want to show in the present section. The particular syntax we use for deductive rules is not important, because we will only use features supported by any reasonable DOOD system. Each rule we will consider deduces facts of some intensionally defined unary predicate (or set). These predicates will represent the attributes of the BAG; a fact $a(\mathbf{n})$ will represent that the attribute value for a is true in node $\mathbf{n}$. We will always write strongly typed rules: all variables occurring in a rule will be explicitly declared to range over all and only objects of a specified class (and its subclasses). Thereto, we will use the following syntax:

$$\textbf{var} \quad \langle\text{variable declarations}\rangle \quad \textbf{in} \quad \langle\text{deductive rules}\rangle.$$

Finally, we note that if x is a variable, then $x.i$ will denote the ith component of the tuple value of the object for which x stands.

For simple BAGs, such as the one of Example 5, the translation to a set of deductive rule is straightforward. Every semantic rule of the BAG is translated into a deductive rule in the obvious manner. For example, the first semantic rule of the BAG of Example 5 is translated into the deductive rule

var $x : \text{HP} \to \text{Name Personal Publs}$
$\quad y : \text{Publs}$
in $\mathit{Jones}(y) \leftarrow x.1 = \texttt{'Jones'}, x.3 = y$

and the second into

var $x : \text{Publs} \to \text{Pub Publs}$
$\quad y : \text{Pub}$
in $\mathit{Jones}(y) \leftarrow \mathit{Jones}(x), x.1 = y.$

It is intuitively clear that the deductive rule program thus obtained correctly computes the query expressed by the BAG.

Generalizing from this case study, we formally describe the following:

Naive Translation Algorithm. Let $a(i) := \varphi$ be a semantic rule of a BAG, in some context (p, a, i), with $p : X_0 \to X_1 \dots X_n$. Assume, without loss of generality, that φ is in disjunctive normal form $\gamma_1 \vee \dots \vee \gamma_m$. Then for this semantic rule we generate, for $\ell = 1, \dots, m$, the deductive rules

var $x_0 : X_0 \to X_1 \dots X_n$
$\quad x_j : X_j$ (for $j = 1, \dots, n$)
in $a(x_i) \leftarrow x_1 = x_0.1, \dots, x_n = x_0.n, \tilde{\gamma}_\ell$

[4] For inexpressibility results on SQL-like query languages we refer the reader to Libkin and Wong [12].

where $\tilde{\gamma}_\ell$ is obtained from γ_ℓ simply by replacing every proposition $b(j)$ occurring in γ_ℓ by $b(x_j)$, and replacing every built-in predicate $f(j_1, \ldots, j_k)$ occurring in γ_ℓ by $f(x_{j_1}, \ldots, x_{j_k})$.

The *naive translation of a BAG* is the deductive rule program consisting of the deductive rules generated from the semantic rules of the BAG. □

From the moment the defining formulas of the semantic rules use negation in an essential way, negation will also occur in the deductive rules, and it is not a priori clear which semantics should be used.

An example of a BAG involving negation is the following one over our example schema, which expresses the query "give all publications not co-authored by Jones":

$$\begin{array}{ll} \text{Pub} \to \text{Title Authors Journal Year} & \mathbf{select}(0) := \neg \mathit{Jones}(2) \\ \text{Authors} \to \text{Name} & \mathit{Jones}(0) := (1 = \texttt{`Jones'}) \\ \text{Authors} \to \text{Name Authors} & \mathit{Jones}(0) := (1 = \texttt{`Jones'}) \vee \mathit{Jones}(2) \end{array}$$

Naively translating this BAG into deductive rules yields:[5]

var x : Pub → Title Authors Journal Year
y : Authors
u : Authors → Name
v : Authors → Name Authors
in $\mathbf{select}(x) \leftarrow y = x.2, \neg \mathit{Jones}(y)$
$\mathit{Jones}(u) \leftarrow u.1 = \texttt{`Jones'}$
$\mathit{Jones}(v) \leftarrow v.1 = \texttt{`Jones'}$
$\mathit{Jones}(v) \leftarrow v.2 = y, \mathit{Jones}(y)$

Which semantics is the appropriate one for this program? In this case it is clear that the program is stratifiable, and that the stratified semantics is the appropriate one.

However, stratification is not always possible. Indeed, BAGs may well involve recursion through negation, as shown by the following simple example, which selects every other homepage from a list of homepages:

$$\begin{array}{ll} \text{HPs} \to \text{HP HPs} & \mathbf{select}(0) := \neg \mathbf{select}(2) \\ \text{HPs} \to \varepsilon & \mathbf{select}(0) := \text{true} \end{array}$$

Naive translation yields the following unstratifiable program

var x : HPs → HP HPs
y : HPs
in $\mathbf{select}(x) \leftarrow y = x.2, \neg \mathbf{select}(y)$

var x : HPs → ε
in $\mathbf{select}(x) \leftarrow$ true.

[5] We have grouped the generated rules and renamed the variables for improved clarity.

It is well known that on this program the well-founded semantics gives the desired result (e.g., [4]). We now prove that in general the well-founded semantics on the naive translation of any BAG captures the evaluation of this BAG. This is another illustration of the naturalness of the well-founded semantics.

Theorem 6. *Let $\mathcal{B}$ be a BAG, P the naive translation of $\mathcal{B}$ and $\mathbf{T}$ an input. Denote the well-foundend model of P on input T by $P_{\mathrm{wf}}(\mathbf{T})$. Then*

1. *$P_{\mathrm{wf}}(\mathbf{T})$ is total;*
2. *for any attribute a of $\mathcal{B}$ and any node $\mathbf{n}$ of $\mathbf{T}$ that has a as an inherited or synthesized attribute*
   - *the fact $a(\mathbf{n})$ is in $P_{\mathrm{wf}}(\mathbf{T})$ if and only if the attribute value of a for $\mathbf{n}$ is true in the evaluation of $\mathcal{B}$ on $\mathbf{T}$;*
   - *the fact $\neg a(\mathbf{n})$ is in $P_{\mathrm{wf}}(\mathbf{T})$ if and only if the attribute value of a for $\mathbf{n}$ is false in the evaluation of $\mathcal{B}$ on $\mathbf{T}$.*

Proof. (sketch) The evaluation process of BAGs is defined in a non-deterministic way. We make it deterministic by evaluating in each step in parallel all attributes that can be evaluated. Define then, for each $j \geq 0$, $\mathcal{B}_j$ as the result of the evaluation of $\mathcal{B}$ on $\mathbf{T}$ after j or fewer steps. Thus $\mathcal{B}_j \subseteq \mathcal{B}_{j+1}$. We can see the attribute-node pairs as facts: If the BAG $\mathcal{B}$ sets $a(\mathbf{n})$ to true in the j-th step of the evaluation, then $\mathcal{B}_j$ will contain the fact $a(\mathbf{n})$. If $a(\mathbf{n})$ is set to false then $\mathcal{B}_j$ will contain the fact $\neg a(\mathbf{n})$. If attribute a is undefined for node $\mathbf{n}$ at step j, then $\mathcal{B}_j$ contains neither $a(\mathbf{n})$ nor $\neg a(\mathbf{n})$.

For the well-founded semantics we use the notation and the definitions of Abiteboul, Hull and Vianu's book [2], to which we refer for details and background. There the alternating fixpoint is defined as the sequence

$$\begin{aligned} \mathbf{I}_0 &= \bot \\ \mathbf{I}_{j+1} &= \mathit{lfp}(\mathit{pg}(P, \mathbf{I}_j))(\bot), \end{aligned}$$

where $\bot$ denotes the set of all negative facts, $\mathit{pg}(P, \mathbf{I})$ denotes the positive ground version of P given $\mathbf{I}$, and $\mathit{lfp}(P)(\bot)$ denotes the least fixpoint of the (positive) program P starting with $\bot$. We show by induction on j that for all $j \geq 0$,

$$\mathcal{B}_j \subseteq \mathbf{I}_k \qquad \text{for all } k \geq j. \tag{1}$$

The base case, $j = 0$, holds since $\mathcal{B}_0 = \emptyset$. Consider the general case. Suppose $a(\mathbf{n}) \in \mathcal{B}_j$. Let φ be the instantiated formula that defines the attribute a for node $\mathbf{n}$. W.l.o.g. φ is in disjunctive normal form. Let σ be a disjunct that evaluates to true and let S be the set of facts in σ. Then by definition of the BAG evaluation process, $S \subseteq \mathcal{B}_{j-1}$. Let $k \geq j-1$. By induction we have that $S \subseteq \mathcal{B}_{j-1} \subseteq I_k$. Let σ' be the deductive rule correponding to σ. The program $\mathit{pg}(P, I_k)$ then contains the right substitutions for the negated atoms in σ'. Since $\mathbf{I}_{k+1}$ is defined as $\mathit{lfp}(\mathit{pg}(P, \mathbf{I}_k)(\bot))$, and $S \subseteq \mathbf{I}_{k+1}$ (by induction hypothesis), σ' will be evaluated to true in $\mathbf{I}_{k+1}$. Hence $a(\mathbf{n}) \in \mathbf{I}_{k+1}$, as $\mathbf{I}_{k+1}$ is a fixpoint of $\mathit{pg}(P, \mathbf{I}_k)$. Thus $a(\mathbf{n}) \in \mathbf{I}_{k+1}$ for any $k \geq j-1$, i.e., $a(\mathbf{n}) \in \mathbf{I}_k$ for any $k \geq j$. The case $\neg a(\mathbf{n}) \in \mathcal{B}_j$ is dual. This concludes the proof of (1).

Since $\mathcal{B}$ is non-circular, there exists an n such that $\mathcal{B}_n = \mathcal{B}_{n+1}$ and every attribute of every node is defined in $\mathcal{B}_n$. It then follows from (1) that $\mathcal{B}_n = \mathbf{I}_n$ and $\mathbf{I}_n = \mathbf{I}_{n+1}$. The model $\mathbf{I}_n$ is thus the (total) well-founded model and captures the evaluation of $\mathcal{B}$. □

An interesting observation to be made is that a much less sophisticated semantics for deductive rules, the naive bottom-up fixpoint evaluation, also captures the evaluation of the BAG. The naive bottom-up fixpoint evaluation starts with the empty set and then derives positive facts by destructively iterating the immediate consequence operator assuming all facts derived in the previous iteration to be true and the others to be false. All facts that are in the fixpoint (we will show that this fixpoint always exists) are true, the others are considered to be false.

Theorem 7. *Let $\mathcal{B}$ be a BAG and let P be the naive translation of $\mathcal{B}$ into a deductive rule program. Let $P_{\text{fix}}(\mathbf{T})$ denote the result of the naive bottom-up fixpoint evaluation applied to P on input $\mathbf{T}$.*

1. *The naive bottom-up fixpoint evaluation applied to P on input $\mathbf{T}$ always leads to a fixpoint.*
2. *For any node $\mathbf{n}$ of $\mathbf{T}$ and for any the attribute a such that a is an attribute of $\mathbf{n}$, the fact $a(\mathbf{n})$ is in $P_{\text{fix}}(\mathbf{T})$ if and only if the attribute value of a for $\mathbf{n}$ is true in the evaluation of $\mathcal{B}$ on $\mathbf{T}$.*

Proof. (sketch) If a is an intensional predicate, let a^j denote the value of a after the j-th iteration of P on $\mathbf{T}$. Let $\mathcal{B}_j$ be defined as in the previous proof. One can now show by induction on j that for every $j \geq 0$: if $a(\mathbf{n}) \in \mathcal{B}_j$ then $\mathbf{n} \in a^j$ and if $\neg a(\mathbf{n}) \in \mathcal{B}_j$ then $\mathbf{n} \notin a^j$.

From the non-circularity of $\mathcal{B}$ it follows then that there exists an n such that $\mathcal{B}_n = \mathcal{B}_{n+1}$ and for each node $\mathbf{n}$ all its attributes are defined. It follows that P reaches its fixpoint after n iterations and that this coincides with the result of the BAG. □

A semantics that is simpler than the well-founded semantics, is the inflationary fixpoint semantics [3, 11, 2]. The inflationary fixpoint is computed in the same way as the naive bottom-up fixpoint, only now the immediate consequence operator is applied in an inflationary manner instead of in a destructive manner. We now present a modified translation from BAGs to deductive rules that allows us to use the inflationary fixpoint procedure to generate the desired model.

It is clear that the inflationary semantics will not work for the naive translation of a BAG. See the last example. The reason for this is that the naive translation ignores a crucial aspect of the BAG evaluation process: a semantic rule is evaluated only if all propositions on which its defining formula depends have already been evaluated. This consideration leads to the following modification of the naive algorithm.

Modified Translation Algorithm. For each attribute a of the BAG, we introduce an auxiliary intensional unary predicate *OK*-a. The deductive rule

$$a(x_i) \leftarrow x_1 = x_0.1, \ldots, x_n = x_0.n, \tilde{\gamma}_\ell$$

generated by the naive translation algorithm is modified into

$$a(x_i) \leftarrow x_1 = x_0.1, \ldots, x_n = x_0.n, \tilde{\gamma}_\ell, \mathit{OK}\text{-}a(x_i).$$

Moreover, the following additional deductive rule is generated:

$$\mathit{OK}\text{-}a(x_i) \leftarrow x_1 = x_0.1, \ldots, x_n = x_0.n, \Delta,$$

where Δ denotes the conjunction of the goals *OK*-$b(x_j)$ for each proposition $b(j)$ occurring in the defining formula φ of the semantic rule. □

Example 8. Translating the previous BAG we now obtain the following program:

var x : HPs $\rightarrow$ HP HPs
  y : HPs
in **select**$(x) \leftarrow y = x.2, \neg$**select**$(y)$, *OK*-**select**$(x)$
  OK-**select**$(x) \leftarrow y = x.2$, *OK*-**select**(y)

var x : HPs $\rightarrow \varepsilon$
in **select**$(x) \leftarrow$ *OK*-**select**(x)
  OK-**select**$(x) \leftarrow$.

The naive bottom-up fixpoint evaluation of this program on an instance now follows closely the evaluation of the original BAG on that instance. First, we deduce *OK*-**select**(**n**), where **n** is the lowest HPs-node in the tree. This means that the value **select**(**n**) can be determined. In the second iteration we deduce that this value is true, as well as *OK*-**select**(**p**), where **p** is the parent of **n**. This allows us in the third iteration to deduce that **select**(**p**) is false, as well as *OK*-**select**(**pp**), where **pp** is the parent of **p**. In the fourth iteration we then deduce that **select**(**pp**) is true, and so on. □

The following theorem says that all three considered semantics of the modified translation of a BAG captures the evaluation of the BAG itself.

Theorem 9. *Let $\mathcal{B}$ be a BAG and let P be the modified translation of $\mathcal{B}$ into a deductive rule program. Let $P_{\mathrm{inf}}(\mathbf{T})$ denote the result of the naive bottom-up fixpoint evaluation applied to P on input $\mathbf{T}$.*

1. *For any attribute a of $\mathcal{B}$ and any node $\mathbf{n}$ of $\mathbf{T}$ such that a is an attribute of $\mathbf{n}$, the fact $a(\mathbf{n})$ is in $P_{\mathrm{inf}}(\mathbf{T})$ if and only if the attribute value of a for $\mathbf{n}$ is true in the evaluation of $\mathcal{B}$ on $\mathbf{T}$.*
2. *For any attribute a of $\mathcal{B}$ and any node $\mathbf{n}$ of $\mathbf{T}$ such that a is an attribute of $\mathbf{n}$, the fact $a(\mathbf{n})$ is in $P_{\mathrm{fix}}(\mathbf{T})$ if and only if the attribute value of a for $\mathbf{n}$ is true in the evaluation of $\mathcal{B}$ on $\mathbf{T}$.*

3. *For any attribute a of $\mathcal{B}$ and any node $\mathbf{n}$ of $\mathbf{T}$ that has a as an attribute*
   - *the fact $a(\mathbf{n})$ is in $P_{\mathrm{wf}}(\mathbf{T})$ if and only if the attribute value of a for $\mathbf{n}$ is true in the evaluation of $\mathcal{B}$ on $\mathbf{T}$;*
   - *the fact $\neg a(\mathbf{n})$ is in $P_{\mathrm{wf}}(\mathbf{T})$ if and only if the attribute value of a for $\mathbf{n}$ is false in the evaluation of $\mathcal{B}$ on $\mathbf{T}$.*

Proof. (sketch) Let $\mathcal{B}_j$ be defined as in the proof of Theorem 6.

1. If a is a predicate, then a^j denotes the value of a after the j-th inflationary iteration of the immediate consequence operator associated to P on input $\mathbf{T}$. We claim that for all j: after the j-th iteration, for each attribute a
   (a) the set OK-a^j contains exactly those nodes for which the attribute a is defined in $\mathcal{B}_j$;
   (b) If $a(\mathbf{n}) \in \mathcal{B}_j$ then $\mathbf{n} \in a^{j+1}$, if $\neg a(\mathbf{n}) \in \mathcal{B}_j$ then $\mathbf{n} \notin a^{j+1}$.
   Since $\mathcal{B}$ is non-circular, there exists an n such that $\mathcal{B}_n = \mathcal{B}_{n+1}$ and for each node $\mathbf{n}$ all its attributes are defined. From the above, it then follows that $a(\mathbf{n}) \in \mathcal{B}_n$ if and only if $\mathbf{n} \in a^n$ and $\neg a(\mathbf{n}) \in \mathcal{B}_n$ if and only if $\mathbf{n} \notin a^n$. Thus, P reaches a fixpoint after n iterations and the inflationary fixpoint procedure captures the evaluation of the BAG.
   We now prove the above claim by induction on j. Fix an attribute a. The base case $j = 0$ is trivial. Consider the general case.
   (a) Let $\mathbf{n}$ be a node of $\mathbf{T}$ such that $a(\mathbf{n})$ is defined in $\mathcal{B}_j$ using the instantiated rule φ. By definition of the BAG evaluation process, all facts φ depends on are already defined. Let
   $$S = \{(b, \mathbf{m}) \mid b(k) \text{ or } \neg b(k) \text{ occurs in } \varphi \text{ and node } \mathbf{m} \text{ is substituted for } k \text{ in the evaluation of } \mathcal{B}, \text{ for some } k\}.$$
   By induction we have that for each $(b, \mathbf{m}) \in S$, $\mathbf{m} \in$ OK-b^{j-1}. Then $\mathbf{n} \in$ OK-a^j by definition of the rule defining OK-a.
   The proof of the other direction proceeds in the same way.
   (b) Let $a(\mathbf{n}) \in \mathcal{B}_j$, be defined by the instantiated semantic rule φ in disjunctive normal form. Let σ be a disjunct that evaluates to true, and σ' the deductive rule corresponding to σ in the modified translation. All facts σ depends on are already defined in $\mathcal{B}_{j-1}$, by induction they have the same value in σ' after the j-th iteration. By (1a), $\mathbf{n} \in$ OK-a^j. Thus σ' will evaluate to true in the $(j+1)$-th iteration. In other words, $\mathbf{n} \in a^{j+1}$.
   The case $\neg a(\mathbf{n}) \in \mathcal{B}_j$ is analogous.
2. The proof is exactly the same as for the previous item.
3. The proof is exactly the same as for Theorem 6, if we make the additional observation that OK-$a(\mathbf{n})$ is in $\mathbf{I}_j$, $j \geq 1$, for every attribute a and every node $\mathbf{n}$ that has a as an attribute. □

We can conclude that, although the result of the translation of a BAG into a deductive rule program will generally contain recursion and negation, and even be unstratifiable, no sophisticated semantics (e.g., the well-founded semantics) is needed; the naive bottom-up fixpoint evaluation suffices. Apart of the class of stratifiable programs, we are not aware of any other interesting class of programs reported in the literature that has this property.

6 Discussion

We conclude this paper with a brief discussion of related papers. We do not attempt to give a survey of query languages for structured document databases.

Our approach may be contrasted to that of Christophides et al. [5], in that the approach taken there is to store structured documents in an OODB, as we do, but then to use a standard OO query language (possibly extended with additional features, such as path variables); the original structure of the document given by the original context-free grammar (or SGML DTD) is “forgotten”. In our approach, the query is still expressed directly in terms of the original structure, since a BAG does nothing more than annotating the context-free grammar with semantic rules deriving the attribute values; the OODB system (more specifically, the DOOD system) supporting the structured document query facility provided by the BAGs remains hidden as an underlying invisible query processing engine.

Our work should not be confused with the work by Deransart and Małuszynski [6], which is vaguely similar but entirely different in nature. There, the authors associate to each attribute grammar a logic program such that the set of all possible attributed derivation trees of the attribute grammar coincides with the set of all possible proof trees of the logic program. We are not interested here in generating a derivation tree; in our case the derivation tree is given as the EDB.

Finally, we should mention that perhaps the first application of attribute grammars to database queries and updates was presented by Abiteboul, Cluet and Milo [1], who used attribute grammars with relation-valued attribute values and relational algebra expressions as semantic rules. Since Boolean values are trivial relation values, and propositional logic formulas are very simple relational algebra expressions, our model of BAGs can be thought of as focusing on a specific instance of this approach. On the other hand, inherited attributes were not considered by Abiteboul et al., who were inspired more by compiler-compiler tools like Yacc rather than by general attribute grammars.

In a theoretical study complementing the present paper [13], we show that BAGs express precisely the class of queries that can be specified in monadic second-order logic. At least theoretically, this result indicates that the structured document query facility provided by BAGs strikes the right balance between expressive power and computational complexity. On the one hand, monadic second-order logic can model structured document query languages of the type proposed by Christophides et al., based on a first-order language such as SQL extended with a mechanism to deal with paths in the tree. On the other hand, monadic second-order logic queries on trees can in principle be computed in time linear in the size of the tree.

Acknowledgment

The second author would like to thank Vassilis Christophides, who got him interested in structured document databases and queries.

References

1. S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *Proceedings 19th Conference on VLDB*, pages 73–84, 1993.
2. S. Abiteboul, R. Hull, V. Vianu: *Foundations of Databases*; Addison-Wesley 1995
3. S. Abiteboul and V. Vianu Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43(1):62–124, 1991.
4. N. Bidoit. Negation in rule-based database languages: a survey. *Theoretical Computer Science*, 78:3-83, 1991.
5. V. Christophides et al. From structured documents to novel query facilities. In *Proceedings of the 1994 ACM SIGMOD Internation Conference on Management of Data*, volume 23:2 of *SIGMOD Record*, pages 313–324. ACM Press, 1994.
6. P. Deransart and J. Małuszynski. Relating logic programs and attribute grammars. *Journal of Logic Programming*, 2:119–155, 1985.
7. A. Goldberg and D. Robson. *Smalltalk-80—The Language and its Implementation*. Addison-Wesley, 1985.
8. 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.
9. P. Kanellakis, C. Lécluse, and P. Richard. The O_2 data model. In F. Bancilhon, C. Delobel, and P. Kanellakis, editors, *Building an object-oriented database system: The story of* O_2, chapter 3. Morgan Kaufmann, 1992.
10. 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.
11. Ph. Kolaitis and C. H. Papadimitriou. Why not negation by fixpoint. In *Proceedings 7th ACM Symposium on Principles of Database Systems*, pages 231–239. ACM Press, 1988.
12. L. Libkin and L. Wong. New techniques for studying set languages, bag languages, and aggregate functions. In *Proceedings 13th ACM Symposium on Principles of Database Systems*, pages 155–166. ACM Press, 1994.
13. F. Neven and J. Van den Bussche. On the expressive power of Boolean-valued attribute grammars. Manuscript, 1997.
14. A. Van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620-650, 1991.