Feature Grammars

Albrecht Schmidt*, Menzo Windhouwer, Martin Kersten
(albrecht,windhouw,mk)@cwi.nl

CWI
Kruislaan 413
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

ABSTRACT

We propose a grammatical view of the problem of integrating different data items under a database perspective. We introduce a variant of context-free grammars, called *feature grammars*, whose parsers may rewrite their input stream. This allows us to provide a simple mechanism for describing and maintaining indexes to Internet multimedia documents. Integration of parser instances as mediators into a database system provides a remarkably transparent framework for indexing external sources and also facilitates the use of plug-in modules provided by third parties. Rewriting the input stream allows to (1) interpret input data and replace them by their interpretations, and (2) integrate data from different sources by linking them into the input stream in the spirit of a structuring schema. The techniques described are used in the Dutch Acoi project.

Keywords. Information Systems, Heterogeneous Databases, Information Integration, Context-Free Grammars, Feature Grammars, Multimedia Indexing.

1. INTRODUCTION

In the database literature, the problem of integrating information from heterogeneous sources has received a lot of attention (see [6, 5, 3, 16, 18, 12] for examples). Current approaches to the problem imply either explicit or implicit three tier architectures: wrappers provide access to source data, knowledge-based mediators provide a global schema [13] or reference schema [15] whose instances are then queried as semi-structured views [2] through the query processor of a database. Often techniques taken from rule-based logic programming [15, 14] provide the 'glue' between the wrapper and the mediator. The contribution of this paper is an alternative formulation of this very 'glue' between wrappers and mediators in terms of context-free grammars [9]. Experiences with non-database researchers suggest that a grammatical approach provides a better chance to catch on then a deductive or DBPL approach.

As information integration mainly borrows its techniques from the areas of Artificial Intelligence and Databases [14] a prospective user of the systems must have quite a background in different areas of computer science. In practice, this may be a problem, because researchers in e.g. image analysis have different working habits and methodologies. Therefore it may be worth looking at simpler ways of going about the problem. One of the few concepts known to virtually any computer scientist (and not only to those), is the concept of parsing context-free grammars. Unfortunately, context-free grammars by themselves are quite static and too inflexible to be used in complex tasks. The authors of this paper propose a way of enriching context free grammars and its parsing method, such that they can be sensibly used for information integration. The approach taken is realized and evaluated in the context of Acoi project [1, 10].

The aim of this paper is to show that it is possible to enrich context-free grammars in a simple yet expressive way so that they can play an important role in the integration of data sources.

This paper is organized as follows. In Section 2 we present two examples to highlight the intuition and simplicity of a grammatical approach. Then, in Section 3, we will take a look at the underlying theory and our experimental setup. A small shift of our viewpoint is sufficient to basically regard feature grammars again as context-free grammars. We conclude with project status and future directions of research.

* corresponding author

2. MOTIVATION AND EXAMPLE

In this section, we try to show why we introduce the notion of feature grammars and go through two examples, a real-world and a more abstract example. We also expose some ideas which should elucidate the problems that may occur when we allow certain left-hand grammars symbols, once they fire, to alter the input tape. These special symbols are called detectors. We will also motivate the choice of name in a concrete application scenario.

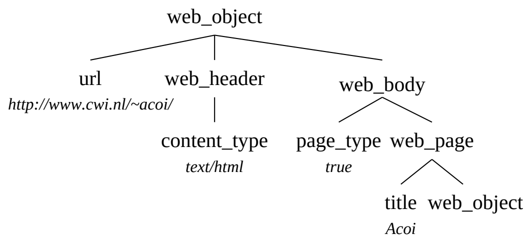
Example 1. We first have a look at a real-world scenario (which is, in fact, very similar to our main application) where we want to index web objects according to their content. In our scenario, we are given several routines, possibly by third parties, to do tasks like image analysis, header analysis and extraction of other features.

An outline of the task ahead runs as follows. We start with an analysis the HTTP header for every web object. Then depending on the content type, feature extractors for images or HTML pages to collect indexing information. This can be accomplished with the following feature grammar:

```
%start       web_object;
%atom        str url,content_type,title;
%detector    web_header(url);
%detector    image_type ?
               content_type="image/gif";
%detector    page_type ?
               content_type="text/html";
%detector    web_page;
web_object: url web_header web_body;
web_header: content_type;
web_body  : image_type web_image
          | page_type web_page;
web_image : ...  /* list of image analysis
                    modules to invoke */
web_page  : ...  /* list of text analysis
                    modules to invoke */
```

The parser associated with this grammar takes `web_object` as the start symbol. In a parse tree representation it forms the root. The symbols `url`, `web_header` and `web_body` are the children of the parent node `web_object`. The leaves of the parse tree atomary lexical items, i.e. `url` is declared as atom. Actually, the url is the only input to the parser; it simply represents an address on the Internet which is then consumed by the `web_header`-detector which gets the document, analyzes its content, and then outputs this information in format readable for the `image_type` and `page_type` detectors. The parsing rules being fired depends on the document content. In the case of an image, processing continues with `web_image`, otherwise with `web_page`. ▪

This schema is suitable for a wide range of applications. However, we also provide a more abstract example to illustrate a range of difficulties we may encounter in certain situations.



Example 2. Language recognition is commonly based on two levels of parsing: lexical analysis and subsequent parsing of a symbol stream. Given an input word $I = \alpha_1\alpha_2\alpha_3\alpha_4$, i.e. the raw input to a lexical analyzer like *lex*, we want to construct a derivation tree. To allow efficient parsing, the first thing to do, is to analyze at least a part of I lexically and map it to symbols at a more abstract level than plain ASCII characters.

The key aspect of traditional parsing technology which we drop is that the lexical analyzer traditionally consumes the head of a primitive token stream. Instead, we use the concept of a detector, which maps $\alpha_1\alpha_2$ to a symbol list $\beta_1\beta_2$, i.e. modifying the input stream to $\beta_1\beta_2\alpha_3\alpha_4$. We use the convention that α_i are raw input data while β_i have undergone a process similar to lexical analysis. The nature of this process may vary from application to application. In the case of a text file it may closely resemble the activities of *lex*, while in the case of an image, a nonlinear analysis could be carried out. We do not restrict the nature of this process. In feature grammars parser and (lexical) analyzer physically operate on the same sequence. As a constraint, raw letters α_i cannot be consumed by the read head, only analyzed β_i can. The parser derived for the feature grammar then merely consumes the head of the modified token stream.

As a next step in our example, the same or another detector might map $\beta_2\alpha_3$ to β_3. The result of this operation would be the symbol sequence $\beta_3\alpha_4$. Here we see a problem: During the parsing process intermediate symbols may come up and disappear again. This poses the questions how to restrict the language of feature grammars. A possible solution to this and related questions will be given below.

The next thing that could happen is the consumption of β_3 by the parser. Parsing may continue in a similar way. ▪

The unpleasant phenomenon of intermediate symbols which appear and disappear again raises the problem of how to define the language of an feature grammar. This and its implications will be dealt with in the following section.

3. ACTIVE GRAMMARS

Let us now turn our attention to the problems mentioned in the example and describe some of the implications. As mentioned, the main underlying problem is that the initial input word $\alpha_1\alpha_2\dots$ is in general not the word actually consumed by the parser. Therefore we should question the traditional definition of the language which is generated by a grammar.

The following two points make the parsing process of feature grammars somewhat different to conventional parsing of 'ordinary' context-free grammars: (1) There is no *a priori* fixed input word. (2) Intermediate symbols may appear and disappear again during parsing and therefore should not be considered part of the language of a feature grammar.

3.1 Theory

We already mentioned that we don't consider it sensible to define the language of a feature grammar G as $L(G) = \{\alpha \,|\, \alpha \text{ can be successfully parsed }\}$ because of the lack of transparency which shows up in intermediate symbols. Furthermore, due to the interference of detectors $L(G)$ may not even be context-free in the traditional sense. We rather want $L(G)$ to reflect the transformations of the input.

The basic idea to cope with these problems is to not regard an input $\alpha_1 \dots \alpha_n$ before parsing as part of the language of a grammar, but the transformed input $\beta_1 \dots \beta_n$, which is the concatenation of the symbols being consumed by the read head. The β_is are the grammar symbols and replace the input sequence. We therefore define as G's language $L(G) = \{\beta \,|\, \beta \text{ has been successfully parsed }\}$.

This way, a plethora of problems caused by intermediate symbols are avoided and we are still able to apply many important theorems from formal language theory. Thus, our next goal is to set up a framework to establish links between context-free grammars and feature grammars. Eventually, we will discover that context-free grammars and feature grammars actually are very similar. In fact, from certain perspectives – most notably when we are interested in the parsing process – we can regard feature grammars as multi-level context-free grammars.

In the sequel, V denotes a set of variables (nonterminals), T a set of terminals, $S \in V$ the start symbol, P a set of productions, D a set of special variables called detectors. G denotes a grammar, Σ the respective input alphabet, We also use the convention that $\alpha_i \in \Sigma$ and $\beta_i \in T$.

Please note that with this set-up, we do not capture all aspects of feature grammars. Especially, detectors are allowed to contact other servers or perform computations which fall out of the framework of parsing technology. To make up for this deficiency, we should consider the introduction of a concept similar to the environment known from modeling methodology of intelligent agents [17]. We will come back to this topic when we give a formal definition of what detectors really are. For the time being, this deficiency is not only accepted but welcomed as it will play a special role when we establish certain properties of common classes of detectors and use them later for query optimization. We also will remark when the given model does not work satisfactorily.

To start with a formalization of our ideas in the usual framework of formal language theory, imagine a nondeterministic stack acceptor [7] which parses an input against a feature grammar. As the tail of the input sequences changes dynamically, we can divide it into two parts: (1) One part is static and already parsed. This part consists of the consumed β_i we mentioned in the Example section. (2) A second part (to the right of the read-head) is possibly rewritten by active nodes/detectors.

In the context of grammars, we normally are not only interested in validating a sequence against a grammar, we also want to construct a derivation tree for a given word $\alpha_1 \dots \alpha_n \in L(G)$. As we use the derivation trees for semantic indexing it is especially interesting to see what properties they have and whether they are actually 'good' representations of the semantics of the indexed objects. This question in particularly interesting in the context of XML documents.

In order to allow efficient parsing, the input must undergo an abstraction procedure; in traditional compiler construction this is called lexical analysis. As pointed out above, we cannot borrow all traditional concepts one-to-one. Therefore we stress that our abstraction is only *similar* to lexical analysis. Abstractions like '3.1415 is a floating point number' or 'what follows is an image with faces' are done by detectors which translate part of the remaining input stream $\alpha_j \dots \alpha_k$ into a stream of recognizable terminal symbols $\beta_j \dots \beta_m$. The resulting stream is consumed by the parser while a tree representation of the derivation process is kept. This structure is basically a traditional parse tree with additional information like time stamps.

To formalize the notion of a detector we must take into account that the behavior of a detector is influenced by the complete input stream and other external factors. Formally, however, it is a mapping from an input sequence $\alpha\beta$ to a sequence β' where α consists of raw and β' of digested (rewritten) input. To capture the behavior of detectors we define the following classes:

Definition. A *context-free detector* for the non-terminal $d \in V$ is a function $d : \Sigma^* \to T^*$. ▪

Definition. A *context-sensitive detector* for the non-terminal $d \in V$ is a function $d : (T^*, \Sigma^*) \to T^*$. ▪

Definition. The actions of a *deterministic* detector depend only on the tokens on the input tape. ▪

Definition. A *safe* detector leaves a parsable token stream behind. ▪

Definition. A detector is called *restricted*, if during parsing all its output resolves to a partial tree whose root is the detector node. ▪

The definitions fall into two classes. The first class, to which context-free, context-sensitive and deterministic detectors belong, captures the different kinds of input to a detector. Note that the problem of intermediate symbols (cf. example at the beginning of this section) does not appear when there are only context-free detectors. In the case of context-sensitive detectors intermediate symbols may be consumed by detectors. In addition, the behavior of deterministic detectors does not only depend on the input on the input tape but also on the 'environment'. While the behavior of context-free detectors is determined only by raw input data, context-sensitive detectors look at a raw and digested input; in addition, detectors which are not deterministic, may also contact the 'environment' (i. e. other servers etc.). Therefore the following inclusions hold:

$$\begin{gathered} D_{\text{context-free}} \\ \subset \\ D_{\text{context-sensitive}} \\ \subset \\ D_{\text{deterministic}} \\ \subset \\ D_{\text{general}}. \end{gathered}$$

Of course, we don't want a detector to output tokens that garble up our parsing process; instead, its output should be a valid input to the parser. This motivates the definitions of safe and restricted detectors. In analogy to the inclusions in the last paragraph, one can easily reason that $D_{\text{restricted}} \subset D_{\text{safe}}$.

Now we come to the central definition of this section. We define:

Definition. We define the language $L(G)$ of an feature grammar G as $L(G) = \{\beta | \exists \alpha : \alpha \overset{*}{\to} \beta\}$ ▪

($\overset{*}{\to}$ has its usual meaning.) Informally, this says that $L(G)$ consists of all β's that have undergone successful parsing. As mentioned above, it is problematic to say that $\alpha = \alpha_1 \dots \alpha_n$ is in $L(G)$ as $\alpha_1 \dots \alpha_n$ is manipulated during the parsing process. To be able to analyze the properties of G we must look at what the read head sees at the time the input is consumed. Actually, this is the word $\beta = \beta_1 \dots \beta_n$. The advantage of the above definitions is that now the concept of detectors fits seamlessly into the theory of context-free grammars as we will see in the following definitions. We recall the following

Definition. A grammar $G = (V, T, S, P)$ is called *context-free* if all productions in P are of the form $A \to x$, where $A \in V, x \in (V \cup T)^*$.

We base our view of feature grammars on the definition of context-free grammars. Regarding detectors as 'active' nodes in the derivation tree under construction, we define:

Definition. We call $G = (V, T, S, P, D)$ a *feature grammar*, if $(V \cup D, T, S, P)$ is a context-free grammar. ▪

Due to the interference of detectors the parse trees of feature grammars have properties which 'normal' trees don't have. One example is that the leaves in feature grammar parse trees are not necessarily labeled with only terminals but with either terminals or detectors. A leaf is labeled with a detector d if d does not rewrite the input string. However, one also expects such a node to have an attribute like a time stamp which indicates when the analysis was made or some synthesized or inherited value.

To be able to talk about partial parse trees, we give the following

Definition. A context-free grammar

$$G' = (V', T', a, P', D')$$

is a minimal sub-grammar of the context-free grammar $G = (V, T, P, S, D)$ if
(1) $(V', T', P', D') \subseteq (V, T, P, D)$,
(2) $a \in V' \cup T' \cup D'$, and
(3) every $V \in V' \cup T' \cup D'$ can be derived from a.

Theorem. Let $G = (V, T, S, P, D)$ be a feature grammar with all detectors $d \in D$ safe and restricted. Then the languages of all of G's minimal sub-grammars with $S \in D$ are context-free. ▪

Sketch of Proof. A first observation is that all rules of the sub-grammar, once they are evaluated, are matched against a fixed token stream. Then there are two cases: (1) no detector is the root of a subtree. (2) a single detector is the root subtree. In case (1) parsing is done as with 'normal' context-free grammars. In case (2) the grammar 'under' the detector node is still context-free and can be parsed. Following this reasoning from the leaves of the parse tree up to the top node proves theorem 1. ▪

To be able to compare the behavior of feature grammars to that of context-free grammars, we define

Definition. A grammar G *behaves* like a context-free grammar if it can be parsed like a context-free grammar.

We immediately apply this definition to our context.

Theorem. A feature grammar G behaves like a context-free grammar. ▪

Sketch of Proof. Theorem 2 holds because all sub-grammars of G are context-free and because the class of context-free grammars is closed under concatenation and embedding. To see this, let both $G_1 = (V_1, T_1, S_1, P_1)$ and $G_2 = (V_2, T_2, S_2, P_2)$ be context-free grammars, and (G_1) and $L(G_2)$ their languages. Then $L(G) = \{w_1 w_2 | w_i \in L(G_i)\}$ is described by $G = (V_1 \cup V_2, T_1 \cup T_2, S, P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\})$, which obviously is context-free. So $L(G)$ is context-free. The same reasoning holds for embedding a grammar G_1 in a rule of G_2. ▪

To sum up, we can see that there are many similarities between context-free grammars and feature grammars. Of course, this has both positive and negative effects. On the one hand, we can apply many concepts from formal language theory to feature grammars. On the other hand, we also inherit the complexity of parsing. Parsing times for many applications even deteriorate because, in general, we don't have a look-ahead due to the 'dynamic' nature of the input stream. To overcome some of these difficulties, we might try to restrict ourselves to a certain subtype of context-free grammars that is in common use: Bracketed Grammars [8]. The structure of Bracketed Grammars closely resembles the structure of XML marked up documents. Introducing such a restriction has the disadvantage that we also impose restriction on the structure of the output of detectors.

But we can also learn from the design of XML. As we want to use parse trees as semantic indices, we are not quite content with what context-free grammars offer. The resulting parse trees often don't have an intuitive form; especially lists don't have a natural linear representation. We therefore introduce regular expression right hand sides in the spirit of [11], which allow us for formulate grammars much like XML DTDs.

Example 3. In this example we show some correspondences between XML and feature grammars. The mapping between DTD and grammar is quite intuitive. Compare the XML-like DTD to the example in Section 2.

```
<!DETECTOR web_object
           (url,web_header,web_body)>
<!DETECTOR url>
<!DETECTOR web_header (content_type)>
<!ELEMENT  web_body
           (image_type web_image
           | page_type web_page)>
<!DETECTOR image_type>
<!DETECTOR page_type>
<!ELEMENT  web_image (photo | logo)>
<!ELEMENT  web_page
           (title?,web_object*)> ...
```

It should be added at this point that changing to bracketed grammars does affect the way detectors have to be designed. For example, not only does the detector `web_header` have to separate HTML Header from HTML Body as it has to do in the feature grammar version; it also has to incorporate XML tags (i.e. typed brackets) at the beginning and at the end of the body block. In practice this means that – in the worst case – all of the input has to be read and written back before processing can continue. In real applications the situation should not be that bad as the input of a detector is normally much larger that its output.

Furthermore, the design of detectors becomes much simpler since the implicit structure of XML documents can be exploited. This is also reflected in a feature space. ▪

3.2 Other Features

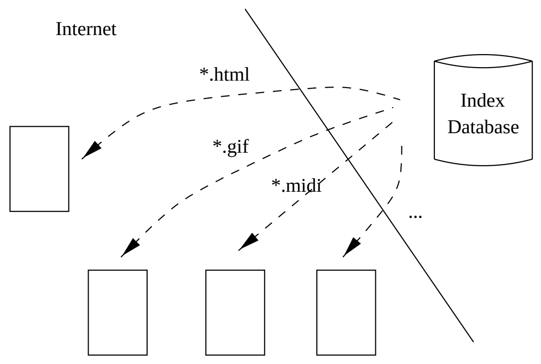
There are several interesting features which we can't describe due to lack of space. Instead, we just mention some interesting points.

Our feature grammar toolkit includes declarative programming language constructs for detectors, which support OQL-like queries on the parse tree under construction and other databases. This leads to an high level declarative definition of a feature space, which is considered a prerequisite for storage and processing optimization.

We should also mention that care has to be taken to ensure that grammars allow efficient parsing; a plethora of work has been done in this area. In particular Bracketed Grammars [8] combined with regular expression-like right hand sides [11] have the advantage of being efficiently parsable and at the same time provide a certain semantic depth. We also stress that feature grammars may simplify integration and view maintenance of XML documents. Of particular interest is the query language mentioned in the preceding section.

3.2 Application

In this section we are going to take a look at the setup of our application. What we describe is implemented in a database system which maintains indexes to data items on the Internet. This setup is depicted in the following figure:



The index database contains URLs of web objects and semantic descriptions in the form of parse trees which were generated by a feature grammar engine. These parse trees contain two kinds of information: administrative and semantic.

Administrative information is expressed by a core feature grammar whose detectors then gather information about modification dates, MIME types and HTML pages containing links to other possible candidates. This administrative information is used to manage the exploration of the web.

For each multimedia type a more specific feature grammar, a subtree of the core grammar, is specified. These grammars lead to semantic descriptions of the specific multimedia type by calling and relating specific detectors. The detectors use algorithms of several multimedia research groups in the Netherlands for e.g. icon generation or language classification.

Queries on the collection of parse trees can now combine context (information from the HTML file in which the multimedia object is embedded), content (basic information extracted from multimedia object) and concept (found by combining both context and content) information. The queries are expressed in a OQL-like query language for which a web interface is currently under development.

Current development also focuses on schema migration and topics like incremental parsing because more and more detectors are become available. This also raises the need for database updates.

4. CONCLUSION

We have presented feature grammars, an extension to context-free grammars specially designed for information integration and maintenance. We think it is a practical and flexible method to integrate barely structured data and plug-in modules in a semi-structured way. We have shown that from a theoretical point of view, they can be seen as context-free grammars with active nodes that may rewrite their input sequence.

The ideas presented are implemented in the Dutch Acoi project [1]. The current status of the project is that a compiler compiler has been implemented; the system is in operation and has been indexing large amounts of web pages, images, and MIDI files using a novel database system [4].

Future work will include topics like incremental parsing of documents, schema transition and further acceleration of the parsing process.

4. REFERENCES

[1] *DMW*. http://www.cwi.nl/~acoi/DMW/.

[2] Serge Abiteboul. Querying semi-structured data. In *ICDT*, pages 1–18, 1997.

[3] K. D. Bollacker, S. Lawrence, and C. L. Giles. Citeseer: An autonomous web agent for automatic retrieval and identification of interesting publications. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–123. ACM Press, May 1998.

[4] P. Boncz and M. L. Kersten. Monet: An impressionist sketch of an advanced database system. In *Proc. IEEE BIWIT Workshop, San Sebastian (Spain)*, July 1995.

[5] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion! In *ACM SIGMOD*, pages 177–188, 1998.

[6] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the world wide web: A survey. *ACM SIGMOD Record*, 27(3):59–74, 1998.

[7] R. W. Floyd and R. Beigel. *The Language of Machines*. Computer Science Press, 1994.

[8] S. Ginsburg and M. A. Harrison. Bracketed context-free grammars. *Journal of Computer and System Sciences*, 1(1):1–23, 1967.

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

[10] M. L. Kersten, N. Nes, and M. Windhouwer. A feature database for multimedia objects. In *ERCIM Database Research Group Workshop on Metadata for Web Databases*, pages 49–57, Bonn, Germany, 1998.

[11] W. R. LaLonde. Regular right part grammars and their parsers. *Communications of the ACM*, 20(10):731–741, 1977.

[12] V. Lattes and M. C. Rousset. The use of carin language and algorithms for information integration: the picsel project. In *Second International and Interdisciplinary Workshop: Intelligent Information Integration*, pages 127–140, 1998.

[13] Juan C. Lavariega and Susan D. Urban. Donají: A semantic architecture for multidatabase systems. In *Second International and Interdisciplinary Workshop: Intelligent Information Integration*, pages 39–53, http://www.informatik.uni-bremen.de/~wache/i3/ws-ecai98/, 1998.

[14] A. Levy and M. C. Rousset. Combining horn rules and description logics in carin. *Artificial Intelligence Journal*, 104, 1998.

[15] Oliver M. Duschka Michael R. Genesereth, Arthur M. Keller. Infomaster: An information integration system. In *Proceedings of 1997 ACM SIGMOD Conference*, 1997.

[16] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 132–141. IEEE Computer Society, 1996.

[17] S. J. Russell and P. Norvig, editors. *Artificial Intelligence : A Modern Approach*. Prentice-Hall, 1995.

[18] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *Proceedings of the 1995 ACM SIGMOD International Conference*, 1995.