Constrained Attribute Grammars for Recognition of Multi-dimensional Objects

Giulia M. Pagallo
Apple Computer, Inc.
One Infinite Loop, MS 302-2IS
Cupertino, CA 95014
E-mail: pagallo@apple.com

Abstract

Handwriting recognition is now a standard feature in many hand-held computers. In most systems, recognition is currently limited to recognition of handwritten text and graphics. However, there is a need to extend recognition to multidimensional domains that are traditionally difficult to input with a keyboard on a desktop computer. In this paper, we address the problem of recognizing multi-dimensional objects by introducing a new class of grammars that we call constrained attribute grammars. In a constrained attribute grammar, semantic information is captured by attributes, while spatial relationships are capture by constraints on the attribute values. In addition, the concepts of keyword and relevance of a keyword are considered to reduce the computational complexity of parsing such grammars. A computationally efficient parsing algorithm based on these concepts is also presented.

Keywords: syntactic pattern recognition, parsing algorith..., mathematical formulas interpretation, spatial objects interpretation.

1 Introduction

In pattern recognition, syntactic methods are preferred over statistical methods whenever it is convenient to represent a pattern as a collection of related sub-patterns. However, strictly syntactic methods for pattern recognition are too limited to capture the full complexity of the sub-patterns in some domains. To overcome this limitation, Fu [5] introduced the concept of attribute grammars for pattern recognition. In an attribute grammar, semantic information about a pattern is added as attributes to the syntactic rules.

The semantic information added to each production rule in a grammar is used either to compute the attributes of a pattern using the attributes of the related sub-patterns, or to indicate the applicability of the rule. The patterns are encoded into one-dimensional strings and are parsed using one of two conventional parsing approaches, top-down or bottom-up. However, these techniques cannot be applied to some structured domains, such as mathematical formulas or organizational charts, since the input cannot be easily encoded into a one dimensional string. Various techniques to overcome limitations to top-down and bottom-up parsing for multi-dimensional domains have been considered [2, 4, 10].

In this paper, we present a new class of grammars for recognizing multi-dimensional objects. The grammars establish a relationship between input symbols and rule symbols for domains where spatial relationships cannot be encoded into one dimension. We call these grammars *constrained attribute grammars (CAG)*. As in an attribute grammar, a constrained attribute grammar establishes a structural relationship among sub-patterns of a pattern by syntactic rules and adds semantic information about the patterns by means of the attributes. In addition, a constrained attribute grammar establishes a correspondence between input symbols and rule symbols by using constraints. Also, in a constrained attribute grammar the input

symbols are divided into two classes, keywords and non-keywords, and a relevance measure is applied to the keyword elements. The relevance measure is domain specific and establishes an order for selecting the keyword elements during parsing.

We also present a computationally efficient algorithm for parsing a CAG. The algorithm takes advantage of the concepts of keyword and relevance measure to heuristically narrow the number of rules that can be expanded at any given point while parsing. In practice, this heuristic often avoids expensive backtracking. We can think of the relevance measure as a way to prioritize keywords, and of keywords as local starting points for parsing. Hence, these two concepts allow us to parse groups of spatially related objects in the order that is determined by the keywords and the relevance measure. In addition, domain dependent knowledge can be encoded into the keywords and the relevance measure. This technique allows us to use the same parsing algorithm with no modifications for different domains. For instance, we have used the same parsing algorithm, with no changes, to parse mathematical expressions and organizational charts.

Similar methods have been considered for recognition of visual languages for on-line systems [2,3,4], however, the concepts of keyword and relevance measure are not considered there. The concepts of keyword and relevance measure permit the encapsulation of domain dependent information so that there is neither a need for domain dependent changes, as in [2], nor for interventions by the designer, as in [3]. Also, several methods have been proposed for off-line recognition of spatial objects such as mathematical expressions and organizational charts [6]. While we have not applied our techniques to off-line recognition, the framework presented here is useful in such applications as well.

We implemented a CAG approach to recognition of multi-dimensional objects as part of a hierarchical recognition system [8]. Basically, in a hierarchical recognition system, several recognizers are organized into a hierarchy where each recognizer at one level generates the input for the recognizers at the next level. The recognizers at the first level use unrecognized information as input, e.g. strokes or bitmap images. In our case, a text or a shape recognizer or both provide the basic input elements for recognition of multi-dimensional objects [7,9].

The remainder of this paper is organized as follows. In Section 2 we present the concept of constrained attribute grammars and contrast them with attribute grammars. In Section 3 we describe a computationally efficient algorithm for parsing CAGs. In Section 4 we present an example of a CAG for a simplified mathematical expression domain and describe the key steps in parsing a given expression. In Section 5 we conclude the paper with observations about CAGs and discuss briefly future directions.

2 Constrained Attribute Grammars

A *constrained attribute grammar* is a seven-tuple $G = (V_n, V_k, V_{nk}, A, P, S, M)$ where V_n is a set of non-terminal symbols, V_k is a set of keyword symbols and V_{nk} is a set of non-keyword symbols. The union of V_k and V_{nk} is the set of terminal symbols. A is a set of attributes so that for each terminal or non-terminal symbol x, $A(x)$ denotes the attribute value of x. An attribute usually describes a spatial characteristic of a symbol. For instance the width of a symbol's bounding box.

P is a set of production rules. Each production rule has a syntactic part, a semantic part, and constraints that establish both a partition on the input symbols and a correspondence between subsets of these symbols and rule symbols. The syntactic part has the form:

$$E \rightarrow E_1 E_2 ... E_n \text{ where } n \text{ is a positive integer,}$$

$E \in V_n$, $E_1 \in V_n \cup V_k$ and $E_i \in V_n \cup V_{nk}$. Hence, each rule has at most one keyword. The requirement that each rule has at most one keyword and that such keyword is listed as the first symbol on the right-hand side helps simplify the implementation of the parsing algorithm. However, this requirement is not a limitation since a rule with more than one keyword can be rewritten as a sequence of rules with at most one keyword, and because rule symbols can be written in any order by using the appropriate constraints. The syntactic and semantic parts of a production rule have the same interpretation as in attribute grammars [5]. The constraint part contains a predicate for each right-hand side symbol of a rule. Each predicate selects a subset of input symbols and implicitly establishes a correspondence between the subset and the corresponding symbol on the right-hand side. Non-null constraints are usually expressed in terms of attributes.

S is, as usual, the start symbol. Finally, M is a mapping from keywords into the real numbers. This mapping is a measure of the relevance of a given keyword when parsing some input. The idea is that keywords are considered in order of relevance when parsing. This provides the parsing algorithm with local starting points and reduces the need for backtracking.

In summary, in relationship to attributed grammars, CAGs add the notion of constraints. A constraint associates input symbols (i.e. sub-patterns) to terminal and non-terminal symbols, and it defines spatial relationships between sub-patterns. Also, a CAG distinguishes some of its terminal elements (keywords) and it provides a relevance measure for them. Distinguishing some elements in the input reduces the computational complexity of the parsing algorithm as we discuss in the next section.

3 Parsing Algorithm

In this section we present the basic recursive function for parsing a CAG where each rule is either a replacement rule or it is a rule with exactly one keyword. A replacement rule is a rule that replaces a non-terminal symbol with another symbol (either a non-terminal or a terminal symbol in the grammar). In practice, this turns out to be the most relevant case since the presence of keywords reduces backtracking. Also, we assume that each keyword is the rightmost element in a rule. While this is not a requirement, it simplifies the presentation and the implementation of the algorithm. In [1] we discuss the algorithm for the general case.

The function **ParseCAG** takes as input a CAG G, a set of symbols R and a non-terminal symbol L. Initially, the non-terminal symbol is the start symbol, and R is the set of input symbols. The function returns *success* if the input symbols form a valid expression, otherwise it returns *failure*. If the function returns *success*, the set R contains a parsed tree for the input symbols.

The function **ParseCAG**(G, R, L) is defined as:

Step 1. (End recursion?) If the set R contains more than one symbol go to Step 2. Otherwise, let E be the symbol in R. If there is no rule of the form $L \rightarrow E$, return *failure*. Otherwise, remove E from R. Form the intermediate symbol L, add L to R and return *success*.

Step 2. (Select keyword) If R does not contain any keyword, return *failure*. Compute the relevance measure for each input keyword in R. Call the most relevant keyword k. Mark all the rules *unused*.

Step 3. (Find rule) Select an unused production rule for which k is the rightmost symbol. If there is no such rule, then return *failure*. Otherwise go to Step 4.

Step 4. (Match symbols) Let $E \rightarrow E_1E_2...E_n$ (where $E_1 = k$) be the syntactic part of the rule selected. Let $C_1...C_n$ be the corresponding constraints. Match the keyword symbol k with E_1. For each symbol E_i, call Vi the subset of R_i that satisfies the constraint C_i, $2 \leq i \leq n$.

Step 5. (Recursive call) For $2 \leq i \leq n$, call **ParseCAG**(G, R_i, E_i). If all calls return *success*, remove all the symbols that matched the rule selected from R, and add R_i to R. If any call returns *failure*, mark the current rule as *used* and go to Step 2.

4 Example

To highlight the main aspects of CAGs we present in this section a simple CAG G for recognizing mathematical expressions with only two operators, addition and division line. For this example, any number, and the operators "+" and "division line" are labeled as terminal symbols. In addition, each operator is labeled as a keyword. The symbols S, E, E_1 and E_2 are non-terminal, S is the start symbol and t is a non-keyword terminal symbol.

For each symbol $x \in G$, its attribute value is the ordered tuple of the coordinates of the corner points of the bounding box of the input symbol corresponding to x. We denote it as $A(x)$. The relevance measure M of a keyword is the width of its bounding box. The production rules are given in Table 1. The semantic part of the rule indicates how to compute the attribute for the left-hand side symbol if the rule applies. In this example, the bounding box for the left-hand side symbol is the same as the one for the right-hand side symbol (Rule 1 and Rule 4), or it is the union of the bounding boxes for the right-hand side symbols. Now let's turn to the constraint part. In Table 1, the constraint part of each rule lists a predicate for each symbol in the right-hand side of the rule. The null symbol ϕ indicates that there is no condition to be tested. For instance, in Rule 2, the first predicate indicates that there is no condition to check for "+", while the other two predicates select the symbols to consider when rewriting E_1 and E_2.

We can write a general grammar for mathematical expression by adding rules to the simple grammar of the example in this section. For example, the multiplication operator can be added by adding a rule similar to Rule 2 where addition is replaced by multiplication in the semantic part of the rule. Other operators can be added in a similar way.

Syntactic Part	Semantic Part	Constraints
1. $S \rightarrow E\|t$	$A(S) = A(E)$ or $A(t)$	ϕ
2. $E \rightarrow +E_1E_2$	$A(E) = A(+) \cup A(E_1) \cup A(E_2)$	ϕ , $A(x)$ is to the right of $A(+)$, and $A(x)$ is is to the left of $A(+)$
3. $E \rightarrow -E_1E_2$	$A(E) = A(-) \cup A(E_1) \cup A(E_2)$	ϕ , $A(x)$ is above $A(+)$, and $A(x)$ is below $A(+)$
4. $E_i \rightarrow E\|t$	$A(E_i) = A(E)$ or $A(t)$	ϕ, for $i = 1,2$

Table 1. Production rules for a CAG for simple mathematical expressions

We now describe the key steps in selecting keywords and applying the constraints when parsing the handwritten expression given in Figure 1.a. For this example, let's assume that the relevance of a keyword is the width of its bounding box.

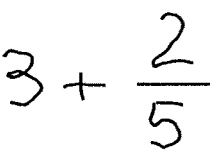

Fig. 1a. Input Expression

Figure 1.b.1 to 1.b.2 show the order in which keywords are selected and how contraints are used to partition the input and associate input elements to non-terminal symbols. Observe that keywords are selected in the following order: first the division line and then the plus sign. Notice that the relevance measure favors the division lines over the plus sign thus avoiding the need for backtraking that would occur if the plus sign was selected first. To complete the parsing successfully, Rule 4 and Rule 1 are applied.

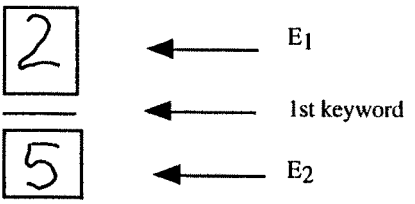

Fig. 1.b.1 First keyword. Boxes show how input elements are associated to the non-terminals in Rule 3 using the rule constraints.

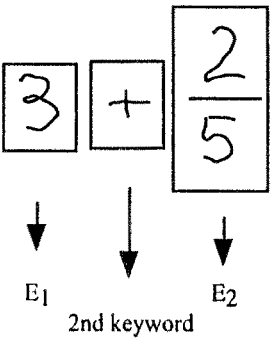

Fig. 1.b.2 Second keyword. Boxes show how input elements are associated to the non-terminals in Rule 2 using the rule constraints.

5 Conclusions

Our initial experiments show that CAG is a promising approach for recognition of spatial objects in an on-line system. The use of keywords and a relevance measure can provide a good heuristic for establishing the order in which the input elements are parsed. The results show that when keywords and a relevance measure are selected appropriately, parsing can proceed with no or little backtracking.

CAG is a flexible approach since the domain dependent information is encoded as parametric data in the rules and keywords. Hence, the description of a domain can be completely captured by the grammar and no changes are required to the parsing algorithm. The modularity of the approach also allows for a concise implementation of several domains within a system. This is an important consideration when designing a solution for hand-held devices since ROM and RAM memory is not readily available.

In the domains we considered, the selection of the keyword elements was quite natural. For instance, each operator is a keyword for mathematical expressions. The relevance measure is a parameter that should be tuned for each domain. It is our experience that even simple measures can give good results. For instance, in the mathematical expressions domain, we started by using the width of a keyword bounding box as a relevance measure.

While we focused on on-line recognition, the techniques presented here could be easily incorporated in an off-line system for document recognition.

References

[1] G. Pagallo "Method And Apparatus For Processing Graphically Input Equations", U.S. Patent 5,544,262, 1996.

[2] R. H. Anderson, "Syntax-Directed Recognition Of Hand-Printed Two-Dimensional Mathematics", in *Interactive Systems for Experimental Applied Mathematics*, M. Klerer and J. Reinfelds, Ed. New York: Academic, 1968.

[3] R. Helm, K. Marriott, and M. Odersky "Building Visual Language Parsers".

[4] A. Belaid and J.P. Haton, "A Syntactic Approach For Handwritten Mathematical Formula Recognition", Vol. Pami-6, No. 1, 105-111, 1984.

[5] K. S. Fu, *Syntactic Methods in Pattern Recognition*, New York: Academic, 1974.

[6] H. Lee and J.S. Wang "Design Of A Mathematical Expression Understanding System", Pattern Recognition Letters 18, 289-298, 1997.

[7] L. Yaeger, R. Lyon, and B. Webb "On-Line Hand-Printing Recognition With Neural Networks", in Proceeding of the Fifth Conference on Microelectronics for Neural Networks and Fuzzy Systems, Lausanne, Switzerland, 1996.

[8] G. Pagallo, E. Beernink, M. T. Tchao, and S. Capps, "Method And Apparatus For Computerized Recognition", U.S. Patent 5,592,566, 1997.

[9] R. Bozinovic and G. Pagallo, "Shape Recognizer For Computer Systems", U.S. Patent 5,544,265, 1996.

[10] R. Mohr, "Precompilation Of Syntactical Descriptions and Knowledge Directed Analysis Of Patterns", Pattern Recognition, Vol. 19, No. 4, pp. 235-256, 1986.