Abstract

With the rapid progress of desktop publishing, documents containing both text and pictures can be easily composed on a computer display. In terms of the internal representation, pictures included in visual documents are usually represented as a sequence of graphic function calls. This paper describes an analysis technique for pictures which are expressed by means of graphic function sequences. This technique has a wide application area such as compilation of visual languages, information retrieval from visual documents, and so forth. We introduce Graphic Functional Grammars in order to specify syntactic structures of pictures. Graphic Functional Grammars are based on Definite Clause Grammar, which has been proposed for describing the syntax of natural languages. A vocabulary consists of not just symbols but graphic functions. A power of describing sentence structures are also enhanced by allowing to add constraints to each production rule. Constraints are written as arithmetic or logical equations and inequations among formal parameters of graphic functions. An efficient parsing algorithm for Graphic Functional Grammars is presented. A novel feature of the parsing algorithm is its order-free property. It can accept any input order of graphic functions as far as they represent a syntactically correct picture. A way of handling graphic functions with side-effect is also discussed.

1. Introduction

With the rapid progress of desktop publishing, documents containing both text and pictures can be easily composed on a computer display. Some researches have concentrated on how to process text included in documents by natural language processing. So far, however, there are few attempts to get informations through analysis of pictures in documents. When we did not have computerized drawing tools, it was difficult to analyze pictures because of the necessity of pattern recognition techniques. Today, most pictures included in visual documents are more easy to deal with, since they are usually stored as a sequence of graphic function calls. For instance, PostScript[7] is widely adopted as a graphic function primitive for many interactive general-purpose drawing tools. Therefore, parsing technique for graphic function sequences should be play a key role in visual situations, just as parsing technique for character sequences does so in textual situation. For example, this technique enables us to compile visual languages, to retrieve various informations from visual documents, and so on[1,2,10,11].

The objective of this paper is to give a formal technique to analyze the syntax of pictures which are represented as graphic function sequences. So far, efforts to analyze the structures of pictures have been made in the context of extending traditional textual languages to multi-dimensional ones[3,4,5,6]. In this context, a picture is considered as scattered symbols which satisfies specific spatial relations in a multi-dimensional space. Although these researches have given a formal framework for visual languages, they do not provide a handy way of processing pictures expressed by means of graphic function calls.

This paper introduces Graphic Functional Grammars (GFG) as an alternative to symbol-based approaches. In Graphic Functional Grammars, a vocabulary consists of not symbols but functions whose parameters are attribute values. For instance, if rectangles are contained as a basic component, we introduce a graphic function drawing a rectangle into its vocabulary, whose parameters may include x-coordinate, y-coordinate, width, height, and so on. Higher level syntactic functions representing more complex visual structures such as graphs and tables are described by a set of production rules generating them. This idea is inspired by Definite Clause Grammar[8] which has been used in natural language processing field. In Definite Clause Grammar, non-terminal symbols are extended to n-ary predicates so that they can include logical variables representing important features such as the number, tense and so forth. Thus, constraints in natural language, such as coincidence of number, can be easily maintained. In Graphic Functional Grammar, spatial constraints are attached to each production rule by means of arithmetic or logical equations and inequations among formal parameters of graphic functions. Graphic Functional Grammar are capable of handling not only spatial attributes but also other visual attributes, both are represented as arguments of graphic functions. In fact, visual attributes such as color, line width, and line type are often as essential as spatial relations such as "above", "on the left", and "in front of". These attributes can be easily dealt with in a consistent and compact way.

This paper is organized as follows. In Section 2 Graphic Functional Grammars is formally defined. Section 3 discusses how to handle side-effects of functions in Graphic Functional Grammars. In Section 4 parsing algorithms for Graphic Functional Grammars are described. Section 5 presents the conclusion.

2. Graphic Functional Grammars

In Graphic Functional Grammars, a sentence of a language is not a sequence of symbols but a sequence of functions. This idea is formally stated as follows.

Definition 2.1 (Graphic Functions)
A graphic function is an n-ary function which draws some geometrical figures in a multi-dimensional space.

Keiji Kojima
Center for Machine Translation
Carnegie Mellon University, Pittsburgh, PA15213, U.S.A.

Brad A. Myers
School of Computer Science
Carnegie Mellon University, Pittsburgh, PA15213, U.S.A.
This definition is somewhat ambiguous. A more precise
definition is given in Section 3.

**Definition 2.2 (Graphic Languages)**

A graphic vocabulary GV is a finite set of graphic functions. A
graphic sentence GS is a finite sequence of graphic functions in
vocabulary, where actual values are assigned to their formal
parameters. A graphic language GL is a set of sentences.

**Example 2.1**

The following function sequence u is a sentence over a
vocabulary V = \{\text{str}(x,y,s), \text{rect}(x,y,w,h)\}).

\[
 u = \text{str}(0,0,\text{"First"}) \text{rect}(0,0,5,3)
\]

The graphic sentence u is shown in Figure 2.1, where \text{str} and
\text{rect} are graphic functions drawing a character string and a
rectangle on a two-dimensional plane.

![Figure 2.1 A simple graphic sentence u.](image)

**Definition 2.3 (Graphic Functional Grammars)**

A graphic functional grammar G is a quadruple (VN, VT, S, P)
where VN is a finite set of non-terminal functions, VT is a
finite set of terminal functions, S is a start function, and P is a
set of production rules. Each production rule in P has the
following form.

\[
\alpha(x_1, \ldots, x_P) \rightarrow \beta_1(y_1, \ldots, y_{I_1}) \cdots \beta_n(y_{N_1}, \ldots, y_{N_q}) \gamma(z_1, \ldots, z_{L})
\]

where \(\alpha(x_1, \ldots, x_P) \in VN\), \(\beta_i(y_1, \ldots, y_{I_i}) \in VN \cup VT\) (\(1 \leq i \leq n\)) and \(\gamma(z_1, \ldots, z_{L})\) is constraints which are denoted
as a formula of first-order predicate logic. An occurrence \(\alpha(t_1, \ldots, t_P)\) in
a graphic sentence can be rewritten to \(\beta_1(t_1,1 \ldots, t_{I_1}) \cdots \beta_n(t_{N_1}, \ldots, t_{N_q}) \gamma(t_1,1 \ldots, t_{L})\) only if \(\gamma(x_1, \ldots, z_{L}) \mid \gamma(t_1,1 \ldots, t_{L})\) is true

A graphic language defined by G is a set of terminal function
sequences which are obtained by repeatedly applying
production rules to the start function.

**Example 2.2**

Let us consider a graphic functional grammar
\(G_{\text{row}} = (VN, VT, S, P)\), where

\[
 VN = \{\text{row}(x,y,h)\},
 VT = \{\text{str}(x,y,s), \text{rect}(x,y,w,h)\},
 S = \text{row}(x,y,h),
 P = \{\text{row}(x,y,h) \rightarrow \epsilon, \text{str}(x,y,s) \text{rect}(x,y,w,h) \rightarrow \text{row}(x+w,y,h) \mid w = \text{length}(s)\}
\]

The language defined by this grammar is a set of rows which
consist of horizontally consecutive rectangular fields
containing strings. For instance, a graphic sentence u in

Example 2.1 belongs to \(GL(G_{\text{row}})\). Figure 2.2 illustrates
the derivation process.

![Figure 2.2 The derivation process for u.](image)

**Example 2.3**

Let us consider another graphic functional grammar
\(G_{\text{column}}\) which is based on the same terminal functions.

\[
 VN = \{\text{column}(x,y,w,h), \text{col}(x,y,w,h,m)\},
 VT = \{\text{str}(x,y,s), \text{rect}(x,y,w,h)\},
 S = \text{column}(x,y,w,h),
 P = \{\text{column}(x,y,w,h) \rightarrow \text{col}(x,y,w,h,m) \mid m = w, \text{col}(x,y,w,h,m) \rightarrow \text{str}(x,y,s) \text{rect}(x,y,w,h) \text{col}(x,y+h,w,h,m,1) \mid m = \text{max(length}(s),m-1)\}
\]

\(GL(G_{\text{column}})\) is a set of columns which consist of vertically
consecutive rectangular fields. Figure 2.3 gives a visual
representation of an example sentence v which belongs to the
language. It should be noted that constraints equate the width
of rectangles to the length of the longest string. Figure 2.4
shows derivation process for v.

**Example 2.4**

The last example in this chapter is a graphic functional
grammar \(G_{\text{graph}}\) which defines a set of directed graphs.

\(G_{\text{graph}}\) is defined as follows.

\[
 VN = \{\text{graph}(p), \text{node}(p), \text{arc}(p)\},
 VT = \{\text{circle}(x,y), \text{arrow}(xs,ys,xe,ye)\},
 S = \text{graph}(p),
 P = \{\text{graph}(p) \rightarrow \text{node}(p) \text{arc}(p), \text{node}(p) \rightarrow \text{circle}(x,y) \text{node}(p) \mid p = p1 \cup \{(x,y)\}, \text{node}(p) \rightarrow \epsilon \mid p = \phi, \text{arc}(p) \rightarrow \text{arrow}(xs,ys,xe,ye) \text{arc}(p) \mid p \supseteq (xs,ys,xe,ye)\}
\]

The terminal function \(\text{circle}(x,y)\) draws a fixed size circle at
\((x,y)\). On the other hand, \(\text{arrow}(xs,ys,xe,ye)\) draws a line with
an arrow which starts at \((xs,ys)\) and ends at \((xe,ye)\). Figure 2.5
The parameter p represents the set of XY-coordinates where nodes are located. It is guaranteed by constraints in the second and third production rules. The constraint in the fourth rule assures that there exist some nodes at both edges of every arrow. Therefore, the following graphic sentence w', which is obtained by removing the circle at (2,2) from w, does not belong to GL(Ggraph).

\[ w' = \text{circle}(1,1) \text{ circle}(4,1) \]
\[ \text{arrow}(1,1,2,2) \text{ arrow}(1,2,2) \text{ arrow}(2,2,4,1). \]

In order to produce circles in w', the parameter p must be \( \{(1,1),(4,1)\} \). The fourth production rule cannot generate arrow(1,1,2,2), however, because its constraint \( \{(1,1),(4,1)\} \not\supseteq \{(1,1),(2,2)\} \) is not satisfied.

The usage of parameter p in above grammar is not so unique. In fact, similar techniques are utilized in natural language processing. As described earlier, Definite Clause Grammars allow to use variables in rules for the purpose of describing the coincidence of the number, the tense, and so on.

### 3. Acceptability of Graphic Sentences

A main problem of graphic functional grammars is the fact that they may not accept graphically valid sentences. For example, let us consider a following graphic sentence w'.

\[ w' = \text{circle}(1,1) \text{ arrow}(1,1,2,2) \text{ arrow}(1,1,4,1) \text{ circle}(2,2) \text{ arrow}(2,2,4,1) \text{ circle}(4,1). \]

It is obvious that w' has the same visual representation as w which is shown in Figure 2.5. The graphic functional grammar Ggraph does not accept w' because it requires circles to precede arrows. One way to resolve this problem is to develop graphic functional grammars which accept all graphically equivalent sentences. It is difficult, however, even for a small example such as Ggraph.

The other approach to this problem is to transform unacceptable but graphically valid sentences into acceptable forms. Such transformations are easy when arbitrary permutations of graphic sentences give a same graphic representation.

#### Definition 3.1 (Commutativity)

A graphic sentence is commutative when its permutations always give a same graphical representation. A graphic functional grammar \( G = (V_N, V_T, S, P) \) is commutative when all sentences in \( GL(G) \) is commutative.

**Example 3.1**

Suppose the display device is monochrome and all graphic functions draw lines in OR-style. Then any grammars on them are commutative.

**Example 3.2**

Let us consider the following grammar Greets where function fillrect fills a rectangle area with a gray tone and setlinewidth sets the line width of successive rectangles.

\[
\begin{align*}
V_N & = \{ \text{Start()} \}, \\
V_T & = \{ \text{fillrect}(x,y,w,h), \text{setlinewidth}(lw) \}, \\
S & = \text{start()}, \\
P & = \{ \text{start()} \rightarrow \text{setlinewidth}(lw) \text{ fillrect}(x,y,w,h) \text{ start()} \\
& \text{ start()} \rightarrow \varepsilon \}.
\end{align*}
\]

As shown in Figure 3.1, Grects is not commutative, because setlinewidth has a side-effect and the order of filling rectangles is significant.
For commutative grammars, we adopt the following as a definition of acceptability.

**Definition 3.2 (Acceptability)**
A graphic sentence $\alpha$ is accepted by a commutative graphic functional grammar $G$ when there exists a permutation $\alpha'$ of $\alpha$ such that $\alpha' \in GL(G)$.

**Example 3.3**
According to Definition 3.3, $w'$ is accepted by $G_{\text{graph}}$, because $w$ is a permutation of $w'$ and $w$ belongs to $GL(G_{\text{graph}})$.

Unfortunately, almost all programming languages for creating graphics do not have commutativity. For example, principal graphic functions of PostScript such as "moveto" have side-effects. Intuitively, side-effects of graphic functions can be eliminated by adding appropriate parameters. In the rest of this section, we discuss a formal way to obtain commutative sentences from non-commutative sentences. First of all, we should extend Definition 2.1 so that it allows to have side-effects.

**Definition 2.1' (Graphic Functions with Side-effects)**
A graphic function $F(x_1, \ldots, x_n)$ is an $n$-ary function, which refers and modifies a finite set of global variables $X_1, \ldots, X_m$. $F$ is decomposed to functions $f_1, \ldots, f_m$ corresponding to write accesses to global variables as follows.

$$F(x_1, \ldots, x_n) \Leftrightarrow$$
$$\begin{align*}
&\text{begin} \\
&\text{for } i = 1 \text{ to } m \text{ do } t_i := X_i; \\
&\text{for } i = 1 \text{ to } m \text{ do } X_i := f_i(x_1, \ldots, x_n, t_1, \ldots, t_m); \\
&\text{end}
\end{align*}$$

The following theorem guarantees that we have only to consider commutative sentences, because non-commutative sentences can be translated into equivalent commutative ones. It should be noted that its proof actually gives an algorithm of the translation.

**Theorem 3.1**
For arbitrary graphic language $L$, there exists a commutative translation from $L$.

**Proof of Theorem 3.1**
Let $V_T$ be a vocabulary of $L$ and $F(x_1, \ldots, x_n)$ be a graphic function which belongs to it. According to Definition 2.1', $F$ is decomposed to function $f_1, \ldots, f_m$ in terms of access to global variables $X_1, \ldots, X_m$. Let us define the following graphic function $\phi_F(x_1, \ldots, x_n, y_1, \ldots, y_m, d)$. We do not mention about drawing facilities in above definition, because we assume that a multi-dimensional geometrical space can be represented by a finite set of global variables such as bitmaps. Under this graphic function model, the equivalence of sentences and vocabularies is defined as follows.

**Definition 3.3 (Equivalence)**
Graphical sentences $\alpha$ and $\alpha'$ are equivalent (denoting $\alpha \equiv \alpha'$) when values of all involved global variables are same after each application.

**Definition 3.4 (Translation)**
A mapping from $L$ to $L'$ is a translation, when all sentences are mapped into equivalent ones. If mapped sentences are always commutative, the translation is called commutative translation.

The following theorem guarantees that we have only to consider commutative sentences, because non-commutative sentences can be translated into equivalent commutative ones. It should be noted that its proof actually gives an algorithm of the translation.

**Theorem 3.1**
For arbitrary graphic language $L$, there exists a commutative translation from $L$.

**Proof of Theorem 3.1**
Let $V_T$ be a vocabulary of $L$ and $F(x_1, \ldots, x_n)$ be a graphic function which belongs to it. According to Definition 2.1', $F$ is decomposed to function $f_1, \ldots, f_m$ in terms of access to global variables $X_1, \ldots, X_m$. Let us define the following graphic function $\phi_F(x_1, \ldots, x_n, y_1, \ldots, y_m, d)$. We do not mention about drawing facilities in above definition, because we assume that a multi-dimensional geometrical space can be represented by a finite set of global variables such as bitmaps. Under this graphic function model, the equivalence of sentences and vocabularies is defined as follows.

**Definition 3.3 (Equivalence)**
Graphical sentences $\alpha$ and $\alpha'$ are equivalent (denoting $\alpha \equiv \alpha'$) when values of all involved global variables are same after each application.

**Definition 3.4 (Translation)**
A mapping from $L$ to $L'$ is a translation, when all sentences are mapped into equivalent ones. If mapped sentences are always commutative, the translation is called commutative translation.

The following theorem guarantees that we have only to consider commutative sentences, because non-commutative sentences can be translated into equivalent commutative ones. It should be noted that its proof actually gives an algorithm of the translation.

**Theorem 3.1**
For arbitrary graphic language $L$, there exists a commutative translation from $L$.

**Proof of Theorem 3.1**
Let $V_T$ be a vocabulary of $L$ and $F(x_1, \ldots, x_n)$ be a graphic function which belongs to it. According to Definition 2.1', $F$ is decomposed to function $f_1, \ldots, f_m$ in terms of access to global variables $X_1, \ldots, X_m$. Let us define the following graphic function $\phi_F(x_1, \ldots, x_n, y_1, \ldots, y_m, d)$. We do not mention about drawing facilities in above definition, because we assume that a multi-dimensional geometrical space can be represented by a finite set of global variables such as bitmaps. Under this graphic function model, the equivalence of sentences and vocabularies is defined as follows.

**Definition 3.3 (Equivalence)**
Graphical sentences $\alpha$ and $\alpha'$ are equivalent (denoting $\alpha \equiv \alpha'$) when values of all involved global variables are same after each application.

**Definition 3.4 (Translation)**
A mapping from $L$ to $L'$ is a translation, when all sentences are mapped into equivalent ones. If mapped sentences are always commutative, the translation is called commutative translation.

The following theorem guarantees that we have only to consider commutative sentences, because non-commutative sentences can be translated into equivalent commutative ones. It should be noted that its proof actually gives an algorithm of the translation.

**Theorem 3.1**
For arbitrary graphic language $L$, there exists a commutative translation from $L$.

**Proof of Theorem 3.1**
Let $V_T$ be a vocabulary of $L$ and $F(x_1, \ldots, x_n)$ be a graphic function which belongs to it. According to Definition 2.1', $F$ is decomposed to function $f_1, \ldots, f_m$ in terms of access to global variables $X_1, \ldots, X_m$. Let us define the following graphic function $\phi_F(x_1, \ldots, x_n, y_1, \ldots, y_m, d)$. We do not mention about drawing facilities in above definition, because we assume that a multi-dimensional geometrical space can be represented by a finite set of global variables such as bitmaps. Under this graphic function model, the equivalence of sentences and vocabularies is defined as follows.

**Definition 3.3 (Equivalence)**
Graphical sentences $\alpha$ and $\alpha'$ are equivalent (denoting $\alpha \equiv \alpha'$) when values of all involved global variables are same after each application.

**Definition 3.4 (Translation)**
A mapping from $L$ to $L'$ is a translation, when all sentences are mapped into equivalent ones. If mapped sentences are always commutative, the translation is called commutative translation.

The following theorem guarantees that we have only to consider commutative sentences, because non-commutative sentences can be translated into equivalent commutative ones. It should be noted that its proof actually gives an algorithm of the translation.
In above definition, \( d_1, \ldots, d_m \) are newly introduced global variables whose initial values are equal to zero. \( V_T' \) is obtained by translating all functions in \( V_T \) according to above method. That is, \( V_T' = \{ \phi \mid \phi \in V_T \} \). And let \( L' \) be a graphic language on \( V_T' \). Then, the following mapping \( cp \) from \( L \) to \( L' \) is a commutative translation.

First, we check the equivalence \( U = cp(u) \in L' \). Because of \( p' = p \) and \( \phi' = \phi \), \( cp(u) \) becomes true. Therefore, according to the definition of \( \phi' \), values of global variables \( X_1, \ldots, X_m \) after the application of \( cp(u) \) are given as follows.

\[
X_i = g_i(p) \quad (0 < i < m + 1).
\]

On the other hand, from the definition of \( g_i(p^{-1}) \), \( g_i(p) \) is equal to the value of \( X_i \) immediately after the application of \( cp(p) \). So according to the definition of \( \phi \), the value of \( X_i \) after the application of \( u \) is equal to \( f_i(p^{-1})(t_i(p^{-1}), \ldots, t_q(p^{-1}), g_i(p^{-1}), \ldots, g_m(p^{-1})) = g_i(p) \). That is, \( u \) and \( \phi(u) \) are equivalent.

Second, let us check the commutativity of \( \phi(u) \). Assume that \( \phi(u) \) is a permutation of \( \phi(u) \). Then, any \( \phi(j) \) which occurs after \( \phi(p) \) does not modify global values because of \( p = d_1(p^{-1}) > d_i(p) = j \). Therefore, the values of global variables \( X_1, \ldots, X_m \) after the application of \( \phi(u) \) are equal to the value after \( \phi(p) \). That is, \( \phi(u) = \phi(u) \). This result shows the commutativity of \( \phi(u) \). Q.E.D.

Example 3.4
Let us make a commutative translation from a language which is shown in Example 3.2. First of all, it is necessary to introduce appropriate global variables. In the case of Example 3.2, the attribute value LINEWIDTH, and pixel values \( Pixel_1, \ldots, Pixel_n \) are involved global variables. \( \text{Setlinewidth} \) and \( \text{Fillrect} \) are obtained as follows.

(1) \( \text{Setlinewidth}(1,0,4,2,1,2) \) \( \Rightarrow \)
\[
\text{begin} \text{dLINEWIDTH} := d; \text{LINEWIDTH} := w; \text{end}
\]
(2) \( \text{Fillrect}(x,y,w,h,l,w,d) \) \( \Rightarrow \)
\[
\text{begin} \text{dPIXEL} := d; \text{PIXEL} := \text{Fillrect}(x,y,w,h,l) \text{end}
\]

In the definition of \( \text{Fillrect} \), \( \text{Fillrect}(x,y,w,h,l) \) is a function which returns the value of \( l \)-th pixel when a specified rectangle is drawn. Now let us check that translated sentences are equivalent and commutative. A sentence \( u1 \) in Example 3.2 is translated into \( \phi(u1) \) where

\[
\phi(u1) = \text{Setlinewidth}(1,0,4,2,1,2) \text{Fillrect}(0,0,4,2,1,2)
\]

In fact, any permutation of \( \phi(u1) \) gives the above picture in Figure 3.1. First, line widths of rectangles do not change because they are explicitly specified as actual parameters. The final value of \( \text{LINEWIDTH} \) is always equal to 6, because \( \text{Setlinewidth}(6,3) \) has the larger second argument value than \( \text{Setlinewidth}(1,1) \). Similarly, \( \text{Fillrect}(3,1,4,2,6,4) \) takes priority to \( \text{Fillrect}(0,0,4,2,1,2) \) in the intersection part of two rectangles. For instance, suppose that \( Pixel_1 \) is the pixel at \( (3,1) \) and \( \text{Fillrect}(3,1,4,2,6,4) \) occurs before \( \text{Fillrect}(0,0,4,2,1,2) \). Then, after applying \( \text{Fillrect}(3,1,4,2,6,4) \), the value of \( Pixel_1 \) and \( Pixel_1 \) are equal to 4 and \( \text{Fillrect}(0,0,4,2,1,2) \). On the other hand, \( \text{Fillrect}(0,0,4,2,1,2) \) is not assigned to \( Pixel_1 \) after applying \( \text{Fillrect}(0,0,4,2,1,2) \) because \( 4 = \text{Fillrect}(0,0,4,2,1,2) \).

It should be noted that this is the same mechanism as Z-buffer algorithm which is used in three dimensional graphics. From this point of view, the final parameter of \( \text{Fillrect} \) can be regarded as a depth in Z-axis. That is, \( \phi \) realizes the commutativity by increasing the dimension of original graphic space.

4. Parsing Algorithms for GFG
Throughout this section, we only discuss commutative grammars. For commutative grammars, as suggested in Definition 3.3, the simplest way to parsing graphic sentences is a generate-and-test method. After generating all possible permutations of a given sentence, each sentence can be tested for acceptability using conventional top-down or bottom-up parsing.

Example 4.1
Graphic sentence \( v \) in Figure 2.3 can be parsed in a top-down manner using appropriate constraints resolving mechanism. Figure 4.1 illustrates the parsing process.

Of course generate-and-test method becomes impractical as a size of a sentence grows because it has an exponential time complexity. Besides, using top-down parsing method, an early evaluation of constraints is difficult because values of variables cannot be instantiated until a parsing process has reached to the bottom of a parse tree. These drawbacks are improved by mixing bottom-up and top-down parsing. The
A resulting constraints are resolved as follows.

\[ \{ m=6 \text{ & } m=\max(4,m1) \text{ & } m1=\max(6,m1') \} \text{ & } m1'=\max(2,m1') \text{ & } m1'=0 \]  
\[ \Rightarrow \text{true.} \]

Figure 4.1 Top-down parsing for \( v \).

Following parsing algorithm for commutative sentences is based on this idea.

**Algorithm 4.1**

**procedure gfgparser(F, C, S, E)**

if \( F \) is the start function and \( S \) is null then return "ACCEPTED."

else begin

**POINT P:** /* develop a tree upward at one level */

find a rule "LHS1 -> RHS1 \( \equiv C1 \)" s.t. RHS1 includes \( F \); if there is no such rule, backtrack to the nearest point; unify \( F \) with RHS1 under environment \( E \) and update \( E \); resolve \( C \& C1 \) under \( E \) and assign the result to \( C \);

**develop a tree downward */

call check_rhs(RHS1, C, S, E);

**develop a tree recursively */

call gfgparser(LHS1, C, S, E);

end

**procedure check_rhs(RHS, C, S, E)**

for all \( F \) in RHS do begin

if \( F \) is terminal then begin

**POINT Q:** remove a function \( G \) from \( S \) s.t. \( F \) matches \( G \); if there is no such function, backtrack to the nearest point; unify \( G \) with RHS under environment \( E \) and update \( E \); resolve \( C \) under \( E \);

end

else begin

**develop a tree downward */

call check_rhs(RHS1, C, S, E);

**develop a tree recursively */

call gfgparser(LHS1, C, S, E);

end

end

In Algorithm 4.1, environment \( E \) means the pairs of variable name and values. To unify means the ordinary unification procedure which is used in logic programming language. Labels are attached to the point which might include some alternative choices. This algorithm first develop a tree in a bottom-up manner at only a single level, then go down as far as possible. When all downward development has been done, again a bottom-up action is taken. We show how the algorithm works thorough an example.

The complexity of Algorithm 4.1 depends on the characteristics of constraints. When the next function choice at POINT \( Q \) is uniquely determined by the corresponding constraint, it can be easily seen that its time complexity is \( O(N^2) \) where \( N \) is the length of an input function sequence. Many visual languages, such as tables, graphs and so forth, come under this classification.

**Example 4.2**

Let us parse the following graphic sentence \( v1 \), using the graphic functional grammar given in Example 2.3. The visual representation of \( v1 \) is identical to that of \( v \), which is shown in Figure 2.3.

\[ v1 = \text{rect}(0,1,6,1) \text{ rect}(0,0,6,2) \text{ str}(0,1,"123456") \]

\[ \text{str}(0,2,"12") \text{ rect}(0,2,6,1) \text{ str}(0,0,1234). \]

It is clear that \( v1 \) cannot be recognized by a simple top-down parser because the sequence has a different order from the given grammar. Notice that Algorithm 4.1 can handle this case since it employs constraint as a guide to the next function to be scanned.

Figure 4.2 shows the growing process of the parse tree.

In STEP 1, the first function \( \text{rect}(0,0,6,1) \) is got from \( v1 \), and unified with the second rule (POINT \( F \)). After the unification, \( x \) and \( y \) must be 0 and 1. That is, the left leaf must be \( \text{str}(0,0,1234) \).

In STEP 2, nonterminal function \( \text{col}(0,1,6,1,m1) \) is replaced in top-down manner. Again only the second rule is applicable (POINT \( R \)). Because \( x \) and \( y \) are 0 and 2, matched leaves \( \text{str}(0,2,"12") \) and \( \text{rect}(0,2,6,1) \) are retrieved from \( v1 \) (POINT \( Q \)). The new condition \( m1=\max(6,1) \) is attached to \( C \).

In STEP 3, nonterminal function \( \text{col}(0,3,6,1,m1) \) is processed. This time the second rule can not be used because there are no function in \( v1 \) satisfying \( x=0 \) and \( y=3 \). So, the third rule is matched and applied (POINT \( Q \)). By adding the corresponding constraint \( m1'=0 \), \( C \) can be completely resolved. It should be noted that the values of \( m \) and \( m1 \) become 6 after resolving \( C \).
In STEP 4, the parse tree is developed upward. The second rule is matched (POINT P), and x and y are both 0 after the unification. Therefore, str(0.0,"1234") and rect(0.0,6,1) are taken as leaves (POINT Q). The constraint C is immediately resolved since all variables are already instantiated.

In STEP 5, the tree grows upward again using the first rule (POINT P), because there remains no function in the sentence vl. The constraint C is satisfied since all variables are already instantiated. An ideal implementation language for above algorithm is constraint logic programming language because it provides not only backtrack mechanisms but also constraint resolving facilities[9].

5. Conclusions

In this paper, we proposed a technique to analyze pictures which are represented as a sequence of graphic function calls.

(1) We have introduced Graphic Functional Grammars for specifying syntactic structures of pictures. It is confirmed that Graphic Functional Grammars describe the structure of pictures in compact way.

(2) Side-effects of graphic functions can be resolved by giving an appropriate translation. A method of getting such translation has been shown.

(3) An efficient parsing algorithm for Graphic Functional Grammars has been presented. The algorithm can recognize most visual structures such as graphs and tables in O(N^2) time. The parsing algorithm has an order-free property. That is, it can accept any order of input as far as they represent a syntactically correct picture.

References