TR-88-19

FORMALIZATION ON VISUAL PROGRAMMING

LANGUAGES
Formalization on Visual Programming Languages

D. H. Lu, M. S. Hwu, and K. Y. Cheng

Department of Computer Science and Information Engineering
National Taiwan University
and
Institute of Information Science Academia
Taipei, Taiwan, R. O. C.

ABSTRACT

The programming technique that utilizes interactive devices for programmer- and nonprogrammer-professional experts to develop their own application programs has been seen in many up-to-date personal computer systems and workstations recently. The distinguished feature of this programming technique is that the designers need only to express their expertise in some sort of visual forms and flows in programming. Nowadays, programming in visual becomes even popular in many applications such as office automation, computer-aided instruction, database interface, computer animation, computer graphics, pattern recognition and cognition, Chinese computer design, morphological language understanding, and artificial intelligence. The purpose of this paper is aiming to a study on this programming technique from formal language theoretical point of view.
I. INTRODUCTION

The programming technique that utilizes certain interactive devices such as screen cursor, joystick, mouse, touch screen, etc. and certain screen operations such as menu, window, form, diagram, icon, etc. has gained widespread popularity in personal computer systems and workstations recently. According to different application purposes, there are a few visual objects which can be described in terms of form-based, diagram-based or icon-based representations. They have been used extensively in a variety of applications such as office information[5, 28, 30, 31, 32], computer-aided instruction[22, 23, 39], computer-aided design[26], database interface[3, 16, 29], computer graphics[9, 10], image processing[12, 33, 36], scene understanding[37], ideographic language understanding[35], ..., etc. This reveals a great portion of human knowledge representation can be expressed visually.

Interestingly enough, most visual applications are based on the visualization of patterns. For example, an office information system is a visualization of office pattern by using forms as visual objects and their handleings as object relations, a Chinese input is a visualization of Chinese character pattern by using radicals as visual objects and spatial relations as object relations, a flow-chart or a diagram is a visualization of graph, ..., etc. From user's point of view, visual programming languages may be viewed as pattern description languages. On the other hand, from system's point of view, the realization of a pattern description is the inference of a grammar that generates the pattern description language. In this paper, we give visual languages a formal definition and discuss some properties of the languages and their programming styles.
II. VISUAL LANGUAGE DEFINITION

In this section, we shall give the visual language a formal definition which is based on the notion of labeled graphs.

[Definition 2.1] The structured graph $S_c(S)$ is a labeled directed graph, i.e., $S_c(S) = (V, E)$, where $V$ and $E$ are sets of labeled nodes and labeled directed edges, respectively.

Each node of a structured graph represents a concept and each directed edge represents a relation. The structured graph is a visualization of a pattern description which may be described by a visual language. Therefore, the visual language may be defined as follows.

[Definition 2.2] A visual language $L_v$ is defined as

$$L_v = \{ x \mid x \text{ is a graph representation which can be generated by a visual grammar } G_v \} ,$$

and the visual grammar $G_v$ is a four tuple

$$G_v = (V_N, V_T, P, S)$$

where $V_N$ is a set of nonterminals (composite visual objects), $V_T$ is a set of terminals (primitive visual objects), $P$ is a set of production rules, and $S$ is a set of starting symbols.
[Definition 2.3] The language \( L_v \) is called a t-based visual language if the representation of visual objects used in the structured graph is t-based.

There are many possible t-based visual languages which have been used in various applications. Among them, form-based[21, 22, 23, 28, 29, 30, 31, 32], diagram-based[1, 2, 8, 11, 34, 38], icon-based[6, 12, 15, 33] visual languages are the most often seen. However, in practice, more than one t-based visual languages may be used simultaneously in the development of an application program[25].

Let \( S \) be an application program with program segments \( \{S_i\} \) then the structured graph \( S_c(S) \) is the visualization of \( S \) and \( S_c(S_i) \) the visualization of each program segment \( S_i \). Using a visual programming language \( L_v \) to describe each \( S_c(S_i) \), a programming synthesizer infers a grammar \( G_i \) or a structure equivalent to \( G_i \) and finally a grammar \( G = G_1 \cup G_2 \cup ... \cup G_n \) and \( G_i \cap G_j = \emptyset \) for \( i \neq j \) to complete the realization of the application program \( S \). The result of this realization is a generated program which can be run on a machine, i.e., the task of the programming synthesizer is to perform grammatical inference and interpret inference results into programs[13, 20].

The process that utilizes visual languages to construct an application program such that the construction result is acceptable by a machine is called visual programming. The environment under which the users can freely develop their application programs by the visual programming technique is a visual programming environment. However, in this paper we shall not go to the detail
description of the visual programming and its environment concerning each mentioned system, instead, we simply describe them from the above defined visual language theoretical point of view.

III. WEB GRAMMAR IN ICON APPLICATIONS

The web grammar proposed by Pfaltz and Rosenfeld[14] can be used to generate sentences whose visualization are directed graphs with symbols at their nodes. In visual application, it is particular useful for the realization of icon applications.

[Definition 3.1] A web grammar is a four tuple

\[ G = ( V_N, V_T, P, S ) \]

where \( V_N \) is a set of nonterminals (composite icons), \( V_T \) a set of terminals (primitive icons), \( S \) a set of initial webs, and \( P \) a set of web productions. A web production is defined as

\[ \alpha \rightarrow \beta, \gamma \]

where \( \alpha \) and \( \beta \) are webs and \( \gamma \) is an embedding of \( \beta \).

Example 3.1. Each Chinese character is a web whose formation can be generated by a web grammar as shown below (Fig. 3.1).
VN = { 腦, 省, 「«」 }

VT = { 月, 女, 田 }

S = { 腦 }

P:

(1) S → 1

(2) B → 1

(3) C → 1

S = 腦
A = 月
B = 省
C = 「«」
D = 田
E = 女
F = 女

Fig. 3.1(b). The web grammar that generates character 腦.
Example 3.2. The act of an icon-based Chinese input is an iconic language. In Tsang-Chi method, 腦 is visualized as a sentence 月女女田 (a sequence of primitive icons). As shown in Fig. 3.1, each initial web (a Chinese character) can be expressed in terms of a derivation diagram in the derivation of the sentence according to some meta rules provided in the input method. It can be shown that the derivation can be generated by a context-free attributed grammar[15], i.e., each production rule of the grammar shown in Fig. 3.2 is associated with a semantic rule (defined in the predefined meta rules).

Semantic rules used

Extc(腦) = Extc(月) · Extc(皆)

(· denodes a concatenate operator)

Extc(皆) = Extc(女) · Extc(田)

Extc(女) = Extc(女) · Extc(女)

Fig. 3.2. The derivation steps of character 腦.
From this observation, we see an immediate visual application is in the design of a Chinese input simulator[24]. Now, the virtual language for the simulator is the visualization of the structured graph for each Chinese character and the evaluation is based on the realization of the visual grammar that generates sentences of the iconic language.

IV. TREE-GRAMMAR IN FORM-BASED APPLICATIONS

Tree grammar and the corresponding tree automata have been studied and applied to syntactic pattern recognition[18, 19]. As will be seen in the following, most form-based visual applications are a visualization of trees.

[Definition 4.1] Let r(a) be the out-degree (or rank) of the labeled node a. Then an expansive tree grammar $G_t = (V, r, P, S)$ has production of the form

$$X_0 \rightarrow x \quad \text{or} \quad X_0 \rightarrow x$$

$$\begin{array}{c}
X_1 \\
X_2 \\
\vdots \\
X_{r(x)}
\end{array}$$

where $x \in V_T$ and $X_0, X_1, ..., X_{r(x)} \in V_N$.

A typical form-based visual application is the QBE( Query-By-Example ) proposed by Zloof[29]. When a user uses QBE to query, update, define, or control a database, he/she uses some skeleton tables to describe the result of what he wants. The commands in each skeleton table and the table itself can be regarded as an application program. The structured graph of the application program has a tree structure with some leaf nodes to represent fields of skeleton table( each field is a primitive visual object ), a root node to represent the name
of a skeleton table, as well as directed edges to represent the descendant relation between the root node and leaf nodes.

Example 4.1. A QBE query as shown in Fig. 4.1 can be expressed as follows (Fig. 4.2).

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ITEM</th>
<th>COLOR</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>P. WHITE</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 4.1. A simple query.

Fig. 4.2. Structured graph of table 'TYPE'.

The visual grammar which generates this structured graph is the following context-free attributed tree grammar[17].
\[ V_N = \{ S, A, B, C \} \]
\[ V_T = \{ \text{TYPE, ITEM, COLOR, SIZE} \} \]
\[ r(\text{TYPE}) = 3, \quad r(\text{ITEM}) = r(\text{COLOR}) = r(\text{SIZE}) = 0 \]

and

\[ P: \]

<table>
<thead>
<tr>
<th>Syntax rules</th>
<th>Semantic rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>S \rightarrow \text{TYPE}</td>
<td>\quad v(S) = \text{AND}(v(A), v(B), v(C))</td>
</tr>
<tr>
<td>\quad A \quad B \quad C</td>
<td></td>
</tr>
<tr>
<td>A \rightarrow \text{ITEM}</td>
<td>\quad v(A) = (v(\text{ITEM}), p(\text{ITEM}))</td>
</tr>
<tr>
<td>B \rightarrow \text{COLOR}</td>
<td>\quad v(B) = (v(\text{COLOR}), p(\text{COLOR}))</td>
</tr>
<tr>
<td>C \rightarrow \text{SIZE}</td>
<td>\quad v(C) = (v(\text{SIZE}), p(\text{SIZE}))</td>
</tr>
</tbody>
</table>

\[ p(\text{ITEM}) = p(\text{SIZE}) = \text{FALSE}, \quad p(\text{COLOR}) = \text{TRUE} \]

where the function \( p \) indicates whether a field is going to be displayed, function \( v \) represents the field value of a tuple and \( \text{AND}(v(A), v(B), v(C)) \) represents a tuple satisfying all constraints of \( v(A), v(B), \) and \( v(C) \). As can be seen, QBE application programs can be regarded as sentences of a visual language generated by a context-free attributed tree grammar. Also, a QBE system can be regarded as a tree automaton that accepts QBE application programs.

Other form-based visual programming systems like QBE/OBE[4,28], FORMAL[30, 31], and VIPS[22, 23] which have been used to develop
application programs in office information, database management, and computer-aided instruction can be similarly regarded as tree automata for the recognition (or acceptance) of user-defined application programs.

V. DIAGRAM-BASED VISUAL LANGUAGES

Diagram-based visual programming languages use the diagrams such as flowchart[2], Nassi-Shneiderman diagram[11, 27], state transition diagram[34], or Petri net[1, 38] as visual objects to develop user-defined application programs. Usually, diagram-based programming is a procedure-wise style which requires users possessed with at least some programming knowledge. For example, Fig. 5.1 shows a flowchart of a binary search program which can be regarded as a structured graph with the flowchart symbols as nodes and the control flow as edges.

![Flowchart Diagram](image)

Figure 5.1. A flowchart.
The realization of the structured graph shown in Fig. 5.2 (part of Fig. 5.1) can be expressed as the following context-sensitive grammar.

\[
\begin{align*}
\alpha B_1 & \rightarrow \alpha x_1 B_2 \\
\beta B_2 & \rightarrow \beta x_2 \text{yes}B_3 \text{no}B_4 \\
\text{yes}B_3 & \rightarrow \text{yes}x_3 \text{arc}1 \\
\text{no}B_4 & \rightarrow \text{no}x_4 \text{arc}2
\end{align*}
\]

where \( \text{yes}, \text{no}, \text{arc}1, \) and \( \text{arc}2 \in V_T; \ B_1, B_2, B_3, \) and \( B_4 \in V_N; \ \alpha, \beta, x_1, x_2, x_3, \) and \( x_4 \in V_T^*. \)

The context-sensitive language generated by the above grammar can be accepted by a transition network shown as follows.
where $A_i$ ($i = 1, 2, 3, \text{ and } 4$) represent the automata which accept program statements.

Other diagrams such as Nassi-Shneiderman diagram, state transition diagram, and Petri-net can be viewed as flow diagram with slight different representations. Hence, the visualization of the diagram is the directed graph and the realization of the directed graph is the context-sensitive graph grammar.

VI. CONCLUSION

In this paper, a formalization on visual programming languages is presented. We show that an icon-based visualization may be realized by a context-free attributed web grammar, a form-based by an attributed tree grammar, and diagram-based by a context-sensitive graph grammar.

REFERENCE


[19] K. S. Fu and B. L. BHARGAVA, "Tree systems for syntactic pattern recognition," *IEEE*


