GIGA: Graphical Definition of Production Rules in a Spatial Parser Generator

Hiroaki Kameyama,[a] Kazuhisa Iizuka[a]
Buntarou Shizuki,[b] Jiro Tanaka[b]

[a]Doctoral Program in Engineering, University of Tsukuba
[b] Institute of Information Science and Electronics, University of Tsukuba
1-1-1 Tennoudai, Tsukuba, Ibaraki, 305-8573, Japan
{kame, iizuka, shizuki, jiro}@iplab.is.tsukuba.ac.jp

ABSTRACT

We previously developed the spatial parser generator Eviss, which automatically generates a spatial parser by defining the grammar of the visual language. In Eviss, we introduced “action” into CMGs in order to express the behavior of visual programming systems. However, in Eviss, the input of the grammar is performed using the text. If we can use a figure for inputting the grammar and edit it directly, the grammar will become easier to understand. Therefore, we have developed the GIGA system to supplant Eviss, the graphical interface for the CMG Input Window has two additional screens, i.e. a pre-action screen and a post-action screen. GIGA outputs the production rule by comparing the two screens. Using GIGA the user can define the grammar easily and can understand the grammar visually.

KEYWORDS

visual language, visual parser, Constraint Multiset Grammars

1. INTRODUCTION

Visual language means a language that uses figures in addition to text. Visual languages are used in various fields, such as ER diagrams and OMT object diagrams. Visual languages have structures like textual languages. Special purpose diagrams such as ER diagrams and OMT object diagrams are often drawn using special-purpose graphic editors. Because general-purpose editors do not know the meaning of the diagrams, semantic relationships between figure elements are not preserved during editing. Creating an analyzer for every application is a difficult and time-consuming task.

Spatial parser generators automatically generate parser of visual languages by providing their grammars. For visual systems, analyzing visual languages is not enough. Actual visual systems must execute statements according to the result of the analysis and redraw the visual sentence preserving the semantic relationships between figure elements.

We previously developed the spatial parser generator Eviss[1][4][5], which automatically generates a spatial parser by defining the grammar of the visual language. In Eviss, we introduced “action” into Constraint Multiset Grammars (CMGs) [2][6] in order to express the behavior of visual programming system. By providing just the grammar and the action of a visual programming system, we can easily describe a visual system. The defined grammar is analyzed by Eviss, which automatically generates the parser.

However, in Eviss, the input of the grammar is performed using text. If all composition elements and action performed are given in textual form, it is difficult to understand what kinds of figures compose the grammar and what happens when the grammar is analyzed. If we can use a figure for inputting the grammar and edit it directly, the grammar will become easier to understand.

2. SPATIAL PARSER GENERATOR

2.1 Extended Constraint Multiset Grammars

We use Extended CMGs [1] in order to define the grammars of visual systems. A CMGs consists of a set of terminal symbols, a set of non-terminal symbols, a distinctive start symbol, and a set of production rules. The terminal and non-terminal symbols have various attributes. The production rules are used to rewrite a multiset of tokens (the instances of the terminal or non-terminal symbols) for a new symbol. The constraints maintain the relationships between the attributes of the tokens.

The production rule of Extended CMGs is defined as follows:

$$\begin{aligned} T(\vec{x}) \ ::= \ & T_1(\vec{x}_1),...,T_n(\vec{x}_n) \ \mathit{where} \\ & \mathit{exist} \ T_1'(\vec{x}_1'),..., T_m'(\vec{x}_m') \\ & \mathit{not\_exist} \ T_1''(\vec{x}_1''),...,T_l''(\vec{x}_l'') \\ & \mathit{where}\, C \ \mathit{and} \ \vec{x} = F \ \mathit{and} \ A \end{aligned}$$

When the attributes of the "normal" tokens $T_1,\dots,T_n$ satisfy the constraint *C*, the tokens $T_1,\dots,T_n$ are rewritten to the non-terminal symbol *T*. The "exist" tokens $T_1',\dots,T_m'$ are needed to recognize *T* and are not rewritten to *T*. If the "not_exist" tokens $T_1'',\dots,T_l''$ satisfy the constraint *C*, the tokens $T_1,\dots,T_n$ are not rewritten to T. Function F has the attributes $x_1,\dots,x_n$ and $x_1',\dots,x_m'$ of the components as arguments, and the return values of the function are given to the non-terminal symbol T as its attribute.

We have extended the original CMGs to include action *A* defined as "script program executed when the production rule is applied." In the Extended CMGs, we can specify arbitrary actions, such as computing values and rewriting figures.

In this paper, we describe the computation tree (Fig. 1), as an example.

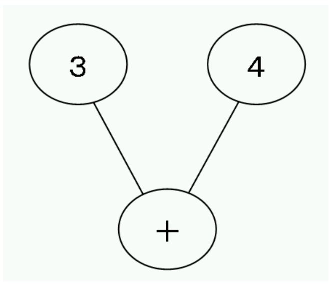


Fig. 1 Computation tree.

The computation tree is defined recursively by the following two production rules.

1. A non-terminal symbol "Node" consists of a circle and a text in the center of it.
2. A non-terminal symbol "Node" consists of a circle, a text string, two nodes and two lines. The two nodes are connected to the circle by the lines.

The production Rule 1 defines a node that represents a number; production Rule 2 defines a node that represents an operator.

These production rules can be written by the Extended CMGs as Fig. 2.

```
 1: Node(point mid, integer value) ::=
 2:       C:Circle, T:Text where (
 3:             C.mid == T.mid
 4:             C.innercolor == "green"
 5:       )   {
 6:               mid = C.mid
 7:               value = T.text
 8:       } and {
 9:       }
10:
11: Node(point mid, string value) ::=
12:       C:Circle, T:Text,
13:       N1:Node, N2:Node,
14:       L1:Line, L2:Line where (
15:             L1.start == N1.mid
16:             L1.end == C.mid
17:             L2.start == N2.mid
18:             L2.end == C.mid
19:             C.mid == T.mid
20:             N1.mid_x < N2.mid_x
21:       )   {
22:               mid = C.mid;
23:               value = { expr
24:               N1.value T.text N2.value }}
25:       } and {
26:               delete { N1,N2,L1,L2 }
27:               alter T.text value
28:       }
```

Fig. 2 Production rule of Extended CMGs.

Lines 1 to 10 define production Rule 1. Line 1 presents the attributes of the non-terminal symbol "Node." Line 2 states the node consists of a circle and a text string. Lines 3 to 4 describe the constraints. Line 3 indicates that the center of the text string is on the center of the circle. Line 4 shows that the inner color of the circle is "green." Lines 6 to 8 define the values of Attributes. Line 6 states "mid" of the "node" is equal to "mid" of "C." Lines 7 to 8 comprise a script for substituting the attribute "text" of "T" for the attribute value of a "node."

Lines 12 to 30 define production Rule 2. Line 12 presents the attributes of the "node." Lines 13 to 15 indicate that the node consists of a circle, a text string, two nodes and two lines. Lines 16 to 22 describe the constraints. Line 16 states that the "start" of "L1" is on the "mid" of "N1." Line 17 states that the "end" of "L1" is on the "mid" of "C." Line 18 states that the "start" of "L2" is on the "mid" of "N2". Line 19 shows that the "end" of "L2" is on the "mid" of "C." Line 20 shows that the "mid" of "C" is on the "mid" of "C." Line 21 shows that the "mid_x" of "N1" is greater than "mid_x" of "C", where "mid_x" represents the x coordinates of point "mid." Line 22 shows that the "mid_x" of "N2" is less than the "mid_x" of "C." Lines 24 to 26 define the values of the attributes. Line 24 shows that attribute “mid” of “node” is equal to the “mid” of “C.” Lines 25 and 26 show that the attribute “value” of the “node” is calculated from the “value” of “N1” and “N2.” Lines 28 to 29 define the action. Line 28 shows that “N1,” “N2,” “L1,” and “L2” are deleted when this production rule is applied. Line 29 shows that the “value” of “T” is replaced by the “value” of “node.”

In Eviss, we provide the following procedures to redraw figures as an action.

- Create: Create a new figure
- Delete: Delete a existing figure
- Alter: Change the attributes (such as colors and fonts) of figure

2.2 Eviss

Fig. 3 depicts an execution snapshot of the Eviss.

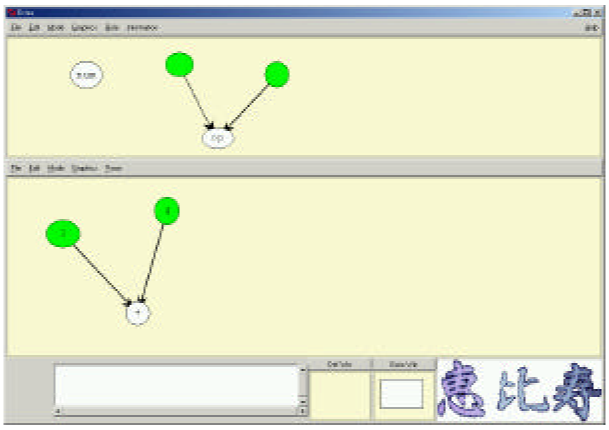


Fig. 3 Execution snapshot of Eviss

The upper half of the screen is called the “definition window.” The user who implements a visual system defines grammar of visual language from the definition window. The bottom half is called the “execution window”. The user draws figure elements to be analyzed in the execution window.

To define the production rules, we use the CMG Input Window (Fig. 4). The CMG Input Window is divided into five parts: Name, Attributes, Action, Constraints and Components. “Components” is divided into “normal,” “exist” and “not_exist.” Name, Attributes, Action and Constraint are written in their respective part. The “normal,” “exist” and “not_exist” components of the new symbol are written in “normal,” “exist,” “not_exist” component part.

In Eviss, rough grammars are first defined using figures. The user draws figures that he wants to define as a new non-terminal symbol from the definition window. We call these “example figures.” Eviss automatically extracts simple constraints and components from each “example figure” and outputs them to the CMG Input Window in textual form. The user then edits the constraints and components in the CMG Input Window. The user can also specify actions and attributes.

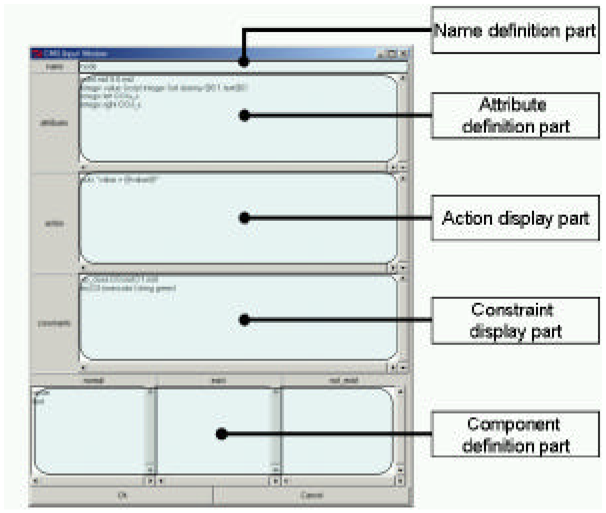


Fig. 4 CMG Input Window

After defining the production rule from the CMG Input Window, the user inputs the figure that he wants to analyze from “execution window.” GIGA then finds the production rules to be applied and rewrites them to the non-teminal symbol. When the input figure is rewritten to the non-terminal symbol, GIGA performs the action defined in the production rule.

3. GRAPHICAL DEFINITION OF RULES

We have newly developed a successor to Eviss called GIGA system. GIGA’s graphical interface for the CMG Input Window has two screens in addition to the CMG input Window of Eviss (as shown in Fig.5). i.e., a pre-action screen and a post-action screen. The user inputs the figures into the Pre-action screen before the action is performed. The user edits the figures in the Post-action screen after the action is performed. Operations in each screen can be performed similarly to a general drawing editor.

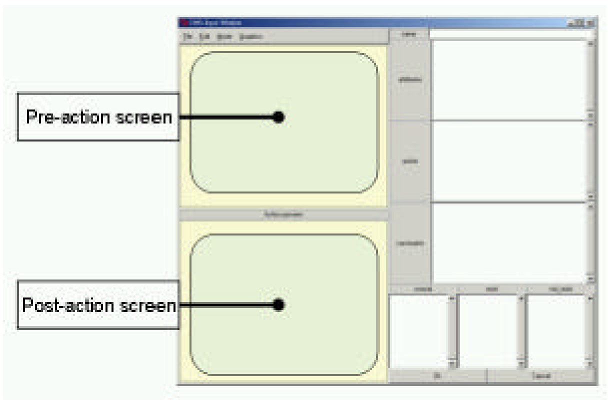


Fig. 5 Graphical interface for CMG input window

The user defines the grammar using the graphical interface for the CMG input window as follows.

1. The user inputs the figures to be analyzed on the Pre-action screen.
2. GIGA duplicates the figure input to the pre-action screen to the post-action screen.
3. On the post-action screen, the user modifies the duplicated figure to show the result after the action is performed.
4. GIGA infers the production rule from two screens and outputs it to the respective definition part in textual form.
5. If required, the textual form can be edited.

3.1 Inference of the component

GIGA infers "normal," "exist" and "not_exist" components by extracting components from the pre-action screen, then outputs them to the Component definition part.

3.2 Inference of the constraint

GIGA infers constraints by extracting them from the pre-action screen, and then outputs them to the Constraint definition part.

3.3 Inference of the attribute

The attribute is calculated using the attribute of the figures that constitute it on the pre-action screen. If the figures have the same attribute, GIGA infers the attribute by synthesizing the same attribute values. If the figures have a different attribute, GIGA infers the attribute by copying the attribute of the figures. GIGA outputs the attributes to the Attribute definition part.

3.4 Inference of the action

When production Rule 2 of the computation tree is applied, the action that should be performed can be defined as follows:

- Two nodes and two lines of a figure that were analyzed by the Rule 2 are deleted, and the operator text in the circle is replaced by the computation result.

In this Example, after the action performed, two nodes and two lines are deleted. Thus, it is possible to infer the action that is executed from the difference of the two pictures.

The kinds of actions, we can use in GIGA, are "create," "delete" and "alter" the attributes. GIGA infers each action by comparing two screens.

- When the figure in the pre-action screen does not exist in the post-action screen (Fig. 6), GIGA generates the "delete" action.

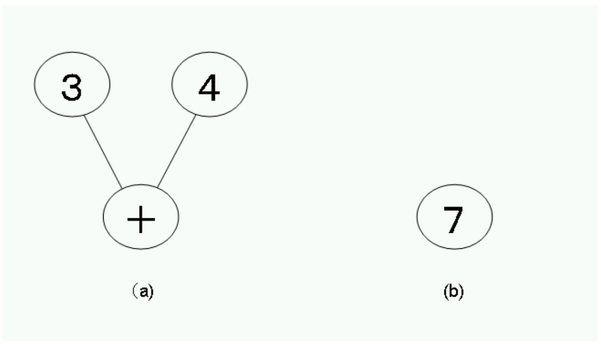


Fig. 6 Action of the computation tree.
(a) before (b) after

- When the attribute of the figure in the pre-action screen is changed on the post-action screen (Fig. 7), GIGA generates the "alter" action.

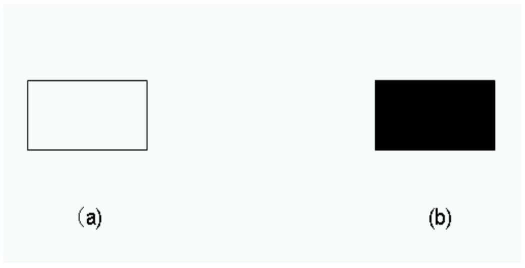


Fig. 7 "Alter" action

- When a figure does not exist in the pre-action screen but exists in the post-action screen (Fig. 8), GIGA generates the "create" action and infers the position in the new figure, making use of the positions of the other figures.

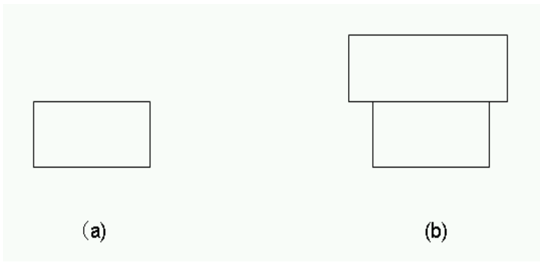


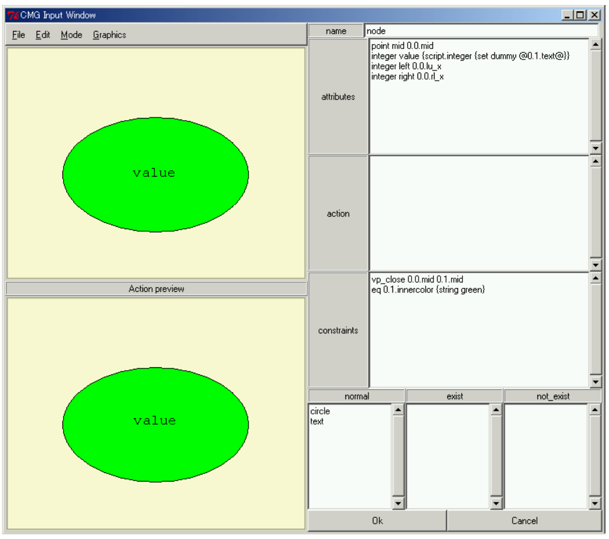
Fig. 8 "Create" action

3.5 Computation tree example

First, we define Rule 1. Since a node consists of a circle and text (number), we draw a circle and text in the pre-action screen. GIGA then infers the grammar of components and outputs it to the Component definition part (shown in Fig. 9).

Since the inner color of the circle is green, GIGA generates the constraint for which the attribute "innercolor" of the circle is equal to "green" and outputs it to the Constraint definition part. Since the center positions of the circle and the text are the same, GIGA generates the constraint in which the attribute "mid" of the circle is equal to the attribute "mid" of the text. GIGA then outputs it to the Constraint definition part.

We then fill the Attribute definition part in textual form.



name: node

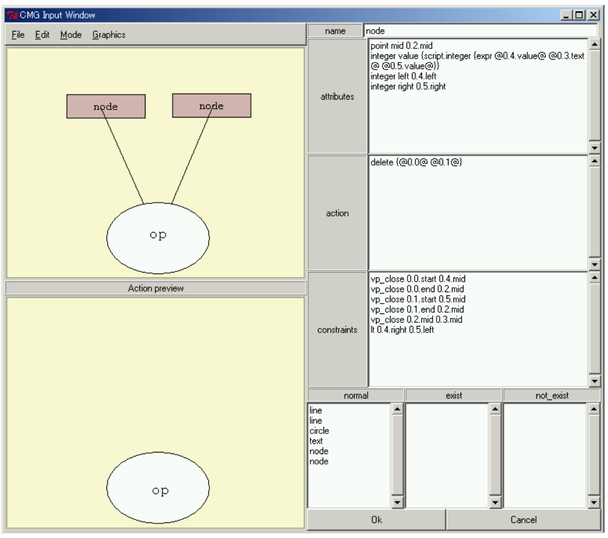
attributes:
```
point mid 0.0.mid
integer value {script.integer {set dummy @0.1.text@}}
integer left 0.0.lu_x
integer right 0.0.rl_x
```

constraints:
```
vp_close 0.0.mid 0.1.mid
eq 0.1.innercolor {string green}
```

normal:
```
circle
text
```

Fig. 9 Definition of production rule 1

Next, we define Rule 2. We draw a circle, a text, two lines and two nodes in the pre-action screen. GIGA then infers the components and outputs them to the Component definition part (shown in Fig. 10). We next delete two nodes and two lines in the post-action screen to define the action of the production rule. GIGA infers the action from the difference of the pre-action screen and the post-action screen, and then outputs it to the Action definition part.



name: node

attributes:
```
point mid 0.2.mid
integer value {script.integer {expr @0.4.value@ @0.3.text
@ @0.5.value@}}
integer left 0.4.left
integer right 0.5.right
```

action:
```
delete {@0.0@ @0.1@}
```

constraints:
```
vp_close 0.0.start 0.4.mid
vp_close 0.0.end 0.2.mid
vp_close 0.1.start 0.5.mid
vp_close 0.1.end 0.2.mid
vp_close 0.2.mid 0.3.mid
lt 0.4.right 0.5.left
```

normal:
```
line
line
circle
text
node
node
```

Fig. 10 The definition of production rule 2

4. RELATED WORKS

KidSim[3] is a visual environment that allows children to create their own simulations. They create their own characters and rules that specify how the characters are to behave. KidSim is programmed by demonstration, so that users do not need to learn a conventional programming language or scripting language. The visual representation for a rule consists of two pictures, i.e. "before" state of the rule and "after" state of the rule.

Visulan[7][8] is a pattern-replacement-rule-based visual language, in which both programs and data are expressed by bitmaps. In Visulan, data processed by programs are pictures called "targets," and programs are the ordered set of pattern-replacing rules called "pairs." Each pair consists of two pictures called a "before-picture" and an "after-picture."

5. CONCLUSION AND FUTURE WORKS

In this paper, we described the GIGA system in which a user can define the grammar using figures. GIGA has two screens for defining the grammar for inferring the production rule from the difference of the two screens. GIGA outputs the resulting production rule by comparing the two screens. With GIGA, the user can define the grammar easily and can understand the grammar visually.

One of our future tasks is to perform the user test and evaluate the validity. We would also like to improve the interface, allowing the user to generate the grammar without using text at all.

REFERENCE

1. Akihiro Baba, Jiro Tanaka, "Eviss: A Visual System Having a Spatial Parser Generator," In *proceedings of 1998 Asia Pacific Computer Human Interaction*, IEEE Computer Society Press, pp. 158-164, Jul 1998.
2. Sitt Sen Chok, Kim Marriott, "Automatic Construction of User Interface from Constraint Multiset Grammars," In *Proceedings of 1995 IEEE Symposium on Visual Languages*,IEEE Computer Society Press, pp. 242-249, Sep 1995
3. Allen Cypher, David Canfield Smith, "KidSim: End User Programming of Simulations," In *Conference proceedings on Human factors in computing systems, Denver, Colorado, United States*, ACM Press/Addison-Wesley Publishing Co, pp. 27-34, May 1995.
4. Kenichiro Fujiyama, Kazuhisa Iizuka and Jiro Tanaka, "VIC:CMG Input System Using Example Figures," In *Proceedings of the international Symposium on Future soft-ware Technology, Najing, China,* pp. 67-72, Oct 1999.
5. Kazuhisa Iizuka, Jiro Tanaka, and Buntarou Shizuki, "Describing a Drawing Editor by Using Constraint Multiset Grammars," In *proceedings of the international Symposium on Future soft-ware Technology, Zhen Zhou China*, pp. 119-124, Nov 2001.
6. Kim Marriot, "Constraint Multset Grammars," In *Proceedings of the 1994 Symposium on Visual Languages,* IEEE Computer Society Press, pp. 118-125, Oct 1994.
7. Kakuya Yamamoto, "3D-visulan: A 3D Programming Language for 3D Applications," In *Proceedings of Pacific Workshop on Distributed Multimedia Systems,* the Hong Kong University of Science and Technology, pp. 199-206, Jun 1996.
8. Kakuya Yamamoto, "Visulan: A Visual Programming Language for Self-Changing Bitmap," In Proceedings of International Conference on Visual Information Systems, Victoria University of Technology (in cooperation with IEEE), pp. 88-96, Feb 1996.