Constraint Relation Multiset Grammars for Visual Languages

福井大学
工学部 研究報告
第45巻 第2号
1997年9月

Constraint Relation Multiset Grammars for Visual Languages

Teruhisa HOCHIN*, Yoshinori ISHII* and Tatsuo TSUJI*

(Received Aug. 29, 1997)

This paper proposes grammars for visual languages. The proposed grammars introduce relations into the constraint multiset grammars[4]. Relations are the constraints on the relationships among symbols in visual languages. The proposed grammars, which are called constraint relation multiset grammars, have the relations as the first class citizens of the grammars because the relationships among symbols play an important role in visual languages. Constraints on both relationships among symbols and attribute values of symbols can be specified in production rules. A parser generator based on the proposed grammars has been constructed.

Key Words : Visual Language, Parser Generator, Constraints, Relationships

1 Introduction

Recently, hardware and software advances enable computer users to use computers in a graphical manner rather than a textual one, e.g. UNIX commands. Many visual languages have been extensively studied and proposed[1] in order to supply the comfortable human-machine interface. Visual languages are the languages that symbols exist in the space of two or more dimensions, and a symbol holds relationships among the others in the space. Visual languages include iconic languages[9, 10, 11, 12, 13], diagrammatic languages[14, 15, 16, 17, 18, 19, 20, 21, 22], three dimensional spatial languages, and two or three dimensional space and (one dimensional) time languages.

It is very hard to code visual language processing system. If the system is written directly in a programming language, the amount of code becomes very large and updating it is very difficult.

* : Department of Information Science

Grammars for defining this kind of language and processing it efficiently are expected, and are extensively investigated. Positional grammars[2], relation grammars[3], constraint multiset grammars[4], and adjacency grammars[5] are such grammars. These grammars extend the grammars for textual languages to be able to be applied to the visual languages.

One of the most distinctive difference between textual and visual languages is that the symbols in a visual language are placed in the space of two or more dimensions, whereas those in a textual one lie in a line. The symbols in a textual language have the relationships of only "preceding" and "succeeding". Each of them is fundamentally related to the symbol preceded or succeeded. On the other hand, a symbol in a visual language relates to other symbols in the space of two or more dimensions. There are various kinds of relationships defined in the space of two or more dimensions. Their examples are left, right, over, under, above, below, crossing, apart, and touching. What symbols are related and how they are related should be naturally and concisely represented in a grammar.

Positional grammars[2] introduce binary relations into context free grammars. Symbols and relations are placed alternately in a line in a production rule. Relative positions of the symbols relating to a relation are specified. Users have to map the relationships in two or more dimensions into those in one dimension. Relation grammars[3] introduce relation symbols and relation productions. The relationships between terminal and/or non-terminal symbols are specified in an *s-production*, which is a production for symbols, through relations. The rule of transformation of a relation is described in an *r-production*, which is a production for a relation. The idea of relation productions is quite interesting. However, it may be difficult to define a grammar of a visual language because r-productions and s-productions are related mutually and deeply. Adjacency constraints are introduced into constraint grammars, e.g. constraint multiset grammars[4], in adjacency grammars[5]. These constraints are good tools in expressing the constraints among adjacent symbols in the multi-dimensional space. It is easy for production rules to be understood. However, it may be too hard to make users use only adjacency constraints.

This paper proposes the grammars introducing *relations* into constraint multiset grammars (CMGs) as the first class citizens of the grammars. *Relations* are the constraints on the relationships among symbols. The proposed grammars are referred to as *constraint relation multiset grammars* (CRMGs). Constraints on the relationships among symbols as well as the attribute values of symbols can be described in CRMGs. This causes several advantages. A conjunction of constraints expressing the relationships among symbols may appear at more than one production rule in CMGs. This conjunction can be described as a relation in CRMGs. A production rule can be described with a relation rather than a conjunction of constraints on the relationship among symbols. Production rules in a grammar can be very concisely described. Names of the relations may help the readers of grammars to understand them well. If the constraints are required to be modified, only the definition of a relation is updated, and production rules relating to it do not have to be updated. The constraints that is neither natural nor easy to be defined as relations can be described as the ordinary constraints in a production rule. A parser generator for visual languages has been constructed by using CRMGs. This generator reads the definition of the symbols, that of the relations, and production rules, and generates a parser processing a visual language.

The remains of this paper is organized as follows: Section 2 describes constraint multiset gram-

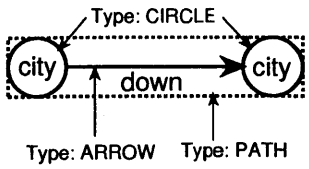

Figure 1: An example of path language

mars, on which the proposed grammars are based. The constraint relation multiset grammars are proposed in Section 3. Section 4 describes the implementation of the parser generator based on the constraint relation multiset grammars. Evaluations on the definition of visual languages are presented in Section 5. Section 6 concludes this paper.

2 Constraint Multiset Grammars

Constraint multiset grammars[4] use constraints over the type symbol attributes to define the relationships between a diagram and its components. Arguments of the constraints are terms built from the computation domain. A constraint multiset grammar specifies a rewriting system over multisets of tokens where a token is a type symbol and an assignment to the attributes of the symbol.

Definition 1 A token, written $T(\vec{\theta})$, consists of a type symbol, T, and a sequence of elements from the computation domain, $\vec{\theta}$, which represents an assignment to the attributes of T. If T is a terminal type, a non-terminal type, or a start type, the token is said to be a terminal token, a non-terminal token, or a start token, respectively. A (multiset) sentence is a multiset of tokens. A terminal sentence is a sentence that contains only terminal tokens.

A constraint multiset grammar is a quadruple (T_T, T_{NT}, S_T, P), where T_T is a set of terminal type symbols, T_{NT} is a set of non-terminal type symbols, S_T is a distinguished start type symbol, and P is a set of productions. Each type symbol $t \in T_T \cup T_{NT}$ has a sequence of attributes. The start symbol may only appear on the left hand side of a production. Productions have the following form: $T(\vec{x}) ::= T_1(\vec{x_1}), \cdots, T_n(\vec{x_n})$ *where exists* $T'_1(\vec{x'_1}), \cdots, T_m(\vec{x_m})$ *where* C *and* $\vec{x} = F$, where T is a non-terminal type symbol, $T_1, \cdots, T_n$ are type symbols with $n \geq 1$, $T_1, \cdots, T_m$ are type symbols with $m \geq 1$, $\vec{x}$, $\vec{x_i}$, and $\vec{x'_i}$ are sequences of distinct variables, C is a conjunction of constraints over $\vec{x_1}, \cdots, \vec{x_n}, \vec{x'_1}, \cdots, \vec{x'_m}$, and F is a function of $\vec{x_1}, \cdots, \vec{x_n}, \vec{x'_1}, \cdots, \vec{x'_m}$. □

Example 1 Consider the grammar that recognizes the diagram in Fig. 1 as a path. The diagram has two labeled nodes (circles) and a directed labeled edge between them. □

The types for this example are `CIRCLE`, `ARROW`, and `PATH`. The type `CIRCLE` has `x_center`, `y_center`, `radius`, and `label` attributes. The type `ARROW` has `x_start`, `y_start`, `x_end`, `y_end`, and `label` attributes. The type `PATH` has `x_upleft`, `y_upleft`, `x_lowright`, and `y_lowright` attributes, where `x_upleft` and `y_upleft` represent the upper left point of minimum bounding box, and `x_lowright` and `y_lowright` represent its lower right point.

Let us consider the production producing a non-terminal symbol of type PATH from two symbols of type CIRCLE and a symbol of type EDGE, of which label is "down". The production is defined as follows.

```
P:PATH ::= C1:CIRCLE, A1:ARROW, C2:CIRCLE
  where
   C1.x_center + C1.radius == A1.x_start
   C1.y_center             == A1.y_start
   C2.x_center - C2.radius == A1.x_end
   C2.y_center             == A1.y_end
   A1.label == "down"
  and
   P.x_upleft   = C1.x_center - C1.radius
   P.y_upleft   = C1.y_center - C1.radius
   P.x_lowright = C2.x_center + C2.radius
   P.y_lowright = C2.y_center + C2.radius
```

Here, P:PATH denotes a symbol P of the type PATH. C1.x_center denotes an attribute x_center of the symbol C1. The first two predicates check the connection of the symbols C1 and A1. The next two predicates check the connection of the symbols A1 and C2. The next predicate A1.label == "down" check whether the label of an edge is "down". Four substitutions, which are described below "and", are for the attribute values of the produced symbol P.

3 Constraint Relation Multiset Grammars

Relations are introduced into CMGs in Constraint Relation Multiset Grammars (CRMGs). CRMGs are formally defined as follows by using the definition of CMGs.

Definition 2 A constraint relation multiset grammar is a quintuple (T_T, T_N, S_T, P, R), where T_T is a set of terminal type symbols, T_N is a set of non-terminal type symbols, S_T is a distinguished start type symbol, P is a set of productions, and R is a set of relations. Each type symbol $t \in T_T \cup T_N$ has a sequence of attributes. The start symbol may only appear on the left hand side of a production. Productions have the same form as those in CMGs except for the constraints C. C is a conjunction of constraints over $\vec{x_1}, \cdots, \vec{x_n}, \vec{x'_1}, \cdots, \vec{x'_m}$, and constraints on relations. Relations have the following form: $rel(T_1(\vec{x_1}), \cdots, T_n(\vec{x_n}))\{C_{rel}, \}$, where C_{rel} is a conjunction of constraints over $\vec{x_1}, \cdots, \vec{x_n}$. □

Here, we consider Example 1 again. Types are the same as those in the CMG described before. Relations among symbols can be described in CRMGs. Here, the relations ConnectS and ConnectE are introduced. The relation ConnectS represents that a symbol of type CIRCLE connects to the starting point of a symbol of type ARROW. The relation ConnectE represents that a symbol of type CIRCLE connects to the terminal point of a symbol of type ARROW. The definition of the relation ConnectS is as follows.

```
relation ConnectS ( C:CIRCLE, A:ARROW )
{
   C.x_center + C.radius == A.x_start
   C.y_center            == A.y_start
}
```

The production rules in producing a non-terminal symbol of type PATH can be described as follows.

```
P:PATH ::= C1:CIRCLE, A1:ARROW, C2:CIRCLE
   where
    ConnectS(C1, A1)
    ConnectE(C2, A1)
    A1.label == "down"
   and
    P.x_upleft   = C1.x_center - C1.radius
    P.y_upleft   = C1.y_center - C1.radius
    P.x_lowright = C2.x_center + C2.radius
    P.y_lowright = C2.y_center + C2.radius
```

As described above, the relations make production rules concise.

Obviously, the following theorem on the expressive power is obtained.

Theorem 1 Constraint relation multiset grammars have the same expressive power as the constraint multiset grammars. □

(Proof) A CMG is the CRMG having the empty set of relations. Contrarily, if a conjunction of the constraints of each relation in a CRMG is directly (and recursively) substituted for the relation, the CRMG becomes the CMG because no relation appears in all of the production rules, and the set of relations may become empty. □

4 Parser Generator

A parser generator NAE (N-dimensional syntax Analysis Environment generator) has been constructed. This parser generator is based on CRMGs. It takes a definition of a visual language, and generates its parser in C++.

It consists of three translators: Type Translator, Relation Translator, and Production Translator, and a common parser (Fig. 2).

Type Translator takes type definition files as input, and generates class definition files in C++, and a class identifier definition file. Types defined in a type definition file are straightforwardly converted to classes in C++. Every type has the member function `Self()` that returns the class identifier. A class identifier is an own number of the class. A class identifier definition file is for defining the class identifiers. The class identifiers are defined through an enumeration type in C++as follows.

```
enum SYMTYPE { RECTANGLE, CIRCLE, ARROW };
```

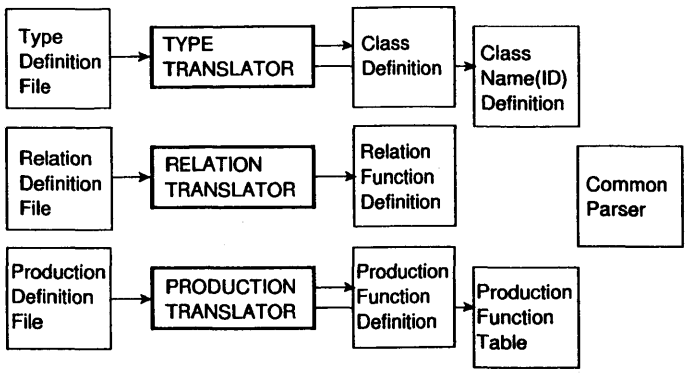

Figure 2: A parser generator NAE

Relation Translator takes relation definition files as input, and generates a relation function definition files in C++. A relation definition is transferred into the C++function that returns a Boolean value.

Production Translator takes production definition files as an input, and generates production function definition files in C++, and a production function table. A production is converted to the C++function that evaluates the constraints, and produces a new non-terminal symbol if the constraints is satisfied. Production Translator puts a name to a production. A production function table manages the function names put to productions, the addresses of the production functions, and the information of their arguments.

Common parser is commonly used by all of the languages. The algorithm in parsing a sentence of this parser is the same as that for a CMG[4]. The parser receives a set of terminal symbols. It applies each production to the possible combinations of the symbols. It returns whether a sentence is received, or not. This algorithm is simple, but is not effective. It should be improved.

From here on, we demonstrate how to use this parser by using an example.

Example 2 Consider the grammar of which terminal symbols are of the type `Rectangle`. If two symbols are horizontal each other, a new symbol of the type `Rectangle` is produced (Fig. 3). □

First, the type definition is shown. The type `Rectangle` is defined as follows.

```
terminal-type Rectangle {
    int  x_upleft;
    int  y_upleft;
    int  x_lowright;
    int  y_lowright;
    String  label;
}
```

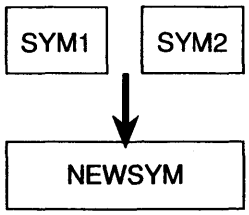

Figure 3: An example of simple visual language

This type is for managing a labeled rectangle. It has x_upleft, y_upleft, x_lowright, y_lowright, and label attributes as the type PATH in Example 1.

Next, a relation is defined by using the attributes of the type defined above. The relation HOR, which evaluates whether two symbols are horizontal, is defined as follows.

```
relation HOR(Rectangle SYM1, Rectangle SYM2 )
{
   SYM1.y_upleft == SYM2.y_upleft;
   SYM1.x_lowright <=  SYM2.x_upleft;
}
```

This relation evaluates two conditions. First, the values of the attribute **y_upleft** of the symbols SYM1 and SYM2. Second, the value of the attribute **x_lowright** of the symbols SYM1 is equal to or less than that of the attribute **x_upleft** of the symbols SYM2. If these two conditions are satisfied, two symbols are determined to be horizontal.

Lastly, a production is defined by using the type and the relation definitions. A production that produces a new symbol of type **Rectangle** from the two horizontal **Rectangle** symbols is defined as follows.

```
NEWSYM:Rectangle ::= SYM1:Rectangle, SYM2:Rectangle
  where
   HOR(SYM1, SYM2);
  and
   NEWSYM.x_upleft = SYM1.x.upleft,
   NEWSYM.y_upleft = SYM1.y_upleft,
   NEWSYM.x_lowright = SYM2.x_lowright,
   NEWSYM.y_lowright = SYM2.y_lowright;
```

Two symbols SYM1 and SYM2 are evaluated whether they satisfy the relation HOR. If it is satisfied, a new symbol NEWSYM is produced. This symbol succeeds the attribute values from those of SYM1 and SYM2.

5 Related Works

Functions can be described in CMGs[4]. Relations in CRMGs correspond to the functions in CMGs. Indeed, relations are converted into functions in C++. However, functions are not the first class citizens in CMGs. They are for the convenience and the adaptability. Relations in CRMGs are the first class citizens in the grammars. This is caused by the fact that relationships among symbols play an important role in visual languages. On this point, CRMGs follow the same approach as adjacency grammars[5].

Adjacencies are introduced in adjacency grammars[5]. These include algebraic adjacency, spatial adjacency, and logical adjacency. Logical adjacency is a broad class of relation. It does not concern the spatial placement of symbols. In this sense, the adjacencies in adjacency grammars are equal to the relations in CRMGs. An adjacency grammar is defined by a quintuple (V_T, V_N, S, P, A), where V_T is a set of terminal symbols, V_N is a set of non-terminal symbols, S is the start symbol, P is a set of productions, and A is a family of adjacency constraints. A production has the following form. $\alpha \rightarrow \{\beta_1, \ldots, \beta_n\}$ *if* $\Gamma(\beta_1, \ldots, \beta_n)$ *after* Δ
where $\alpha \in V_N$, $\beta_j \in V_T \cup V_N$, Γ is a adjacency constraint, and Δ is a set of expressions over attributes of the β_j, which synthesize attributes of α. Δ includes only members of A. In the case of specifying a simple constraint, e.g. `A1.label == "down"`, the constraint has to be included in A. There may be two methods in addressing this issue in the adjacency grammars. First is that adjacency, which is among symbols, is extend to that between a symbol and a constant. For example, adjacency constraint `label( A1, "down")` checking whether the value of the attribute `label` of a symbol `A1` is equal to "down"—. Second is that the adjacency constraint is specialized to be able to also check this kind of simple constraints. In this method, a variety of specialized adjacency constraints may have to be introduced into a family of adjacency constraints A in a grammar. These methods may be cumbersome in the real world applications. The constraints that do not relate relations can be straightforwardly specified in a production in CRMGs. This may be convenient in writing a visual language, which may have a lot of exceptions.

6 Concluding Remarks

This paper proposes constraint relation multiset grammars. The proposed grammars introduce relations into the constraint multiset grammars[4] as the first class citizens of the grammars. Relations are the constraints on the relationships among symbols in visual languages. Visual languages are naturally defined in the proposed grammars because the relationships among symbols play an important role in visual languages. Constraints on relationships among symbols as well as those on attribute values of symbols can be specified in production rules. A parser generator based on the proposed grammars has been constructed. This parser generator takes the type, the relation, and the production definitions as inputs, and produces a parser written in C++.

Improving the parsing algorithm, and applying the parser generator to real applications are the subjects for the future research.

References

[1] Batini, C., Catarci, T., Costabile, M. F. and Levialdi, S., "Visual Query Systems: A Taxonomy," Proceedings of IFIP WG2.6 2nd Working Conference on Visual Database Systems, pp. 159–173 (1991).

[2] G. Costagliola, G. Tortora, S. Orefice, and A. D. Lucia, "Automatic Generation of Visual Programming Environment," COMPUTER, Vol. 28, No. 3, pp. 56-66 (1995).

[3] F. Ferrucci, G. Tortora, M. Tucci, and G. Vitiello, "A Predictive Parser for Visual Languages Specified by Relation Grammars," Proc. of 1994 IEEE Symposium on Visual Languages, pp. 245–252 (1994).

[4] K. Marriott, "Constraint Multiset Grammars," Proc. of 1994 IEEE Symposium on Visual Languages, pp. 118–125 (1994).

[5] J. A. P. Jorge and E. P. Glinert, "Online Parsing of Visual Languages Using Adjacency Grammars," Proc. of 1995 IEEE Symposium on Visual Languages, pp. 250–257 (1995).

[6] J. Rekers and A. Schürr, "A Graph Grammar Approach to Graphical Parsing," Proc. of 1995 IEEE Symposium on Visual Languages, pp. 195–202 (1995).

[7] C. Crimi, A. Guercio, G. Pacini, G. Tortora, and M. Tucci, "Automating Visual Language Generation," IEEE Trans. on Softoware Eng., Vol. 16, No. 10, pp. 1122–1135 (1990).

[8] J. Feder, "Plex Languages," Information Sciences, Vol. 3, pp. 225–241 (1971).

[9] S.-K. Chang, M. J. Tauber, B. Yu, and J.-S. Yu, "A Visual Language Compiler," IEEE Trans. on Softoware Eng., Vol. 15, No. 5, pp. 506–525 (1989).

[10] S.-K. Chang, "A Visual Language Compiler for Information Retrieval by Visual Reasoning," IEEE Trans. on Softoware Eng., Vol. 16, No. 10, pp. 1136–1135 (1990).

[11] M. Hirakawa, M. Tanaka, and T. Ichikawa, "An Iconic Programming System, HI-VISUAL," IEEE Trans. on Softoware Eng., Vol. 16, No. 10, pp. 1178–1184 (1990).

[12] Campbell, D. M., Embley, D. W., and Czejdo, B., " Graphical query formulation for an entity-relationship model," Data & Knowledge Eng., Vol.2, pp.89–121 (1987).

[13] Dart, P. W. and Zobel, J. , " Conceptual Schemas Applied to Deductive Databases," Inform. Systems, Vol.13, No.3, pp.273–287 (1988).

[14] Houchin, T.: "DUO: Graph-based Database Graphical Query Expression," Proc. of 2nd Far-East Workshop on Future Database Systems, pp.286–295 (1992).

[15] Curz, I. F., Mendelzon, A. O. and Wood, P. T., "G^+: Recursive Queries Without Recursion," Proceedings of 12th International Conference on Expert Database Systems, pp. 355–368 (1988).

[16] Consens, M. P. and Mendelzon, A. O., " GraphLog: a Visual Formalism for Real Life Recursion," Proceedings of 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 404–416 (1990).

[17] Paredaens, J., Peelman, P. and Tanca, L., "G-Log: A Declarative Graphical Query Language," Proc. of 2nd International Conference on Deductive and Object-Oriented Databases, pp.108–128 (1991).

[18] Curz, I. F., "DOODLE: A Visual Language for Object-Oriented Databases," Proceedings of 1992 ACM SIGMOD, pp. 71–80 (1992).

[19] Miura, T. , " A Visual Data Manipulation Language for a Semantic data Model," Proc. of COMPSAC91, pp.212–218 (1991).

[20] Auddino, A. et al, "SUPER: A Comprehensive Approach to Database Visual Interfaces," Proc. of IFIP WG2.6 2nd Working Conf. on Visual Database Systems, pp.359–374 (1991).

[21] Angelaccio, M., Catarci, T. and Santucci, G. , " QBD*: A Graphical Query Language with Recursion," IEEE Trans. on Soft. Eng., Vol.16, No.10, pp.1150–1163 (1990).

[22] Schneider, M. and Trepied, C. , " Extensions for the graphical query language CANDID," Proc. of IFIP WG2.6 2nd Working Conf. on Visual Database Systems, pp.189–203 (1991).