ELSEVIER

Information Sciences 131 (2001) 19–46

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

MRG parser for visual languages

Muhammed Al-Mulhem *, Mohammed Ather

Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

Received 10 October 1999; accepted 4 July 2000

Abstract

The theory of visual programming languages (VPLs) is very crucial for understanding the visual programming approach. It includes basically formal models for specifying VPLs and the corresponding parsing algorithms. This paper presents a grammatical formalism and an efficient parsing algorithm for visual languages. The proposed formalism, called modified relation grammar (MRG), is a restricted form of the relation grammar (RG). MRG restricts the form of productions and distributes the evaluation rules among various production rules, which makes the grammar more readable. The paper also presents an efficient O(n) parsing algorithm for MRG. The paper includes a number of examples to demonstrate the proposed formalism. It also includes a discussion on the expressive power of the proposed grammar and the correctness of its parsing algorithm. Finally, the paper compares the proposed grammar and its parsing algorithm with a similar restricted form of the RG. © 2001 Elsevier Science Inc. All rights reserved.

Keywords: Grammar; Parsing; Nonlinear languages; Visual languages; Visual programming

1. Introduction

Visual programming languages (VPL) [4,15,16,19,21,22] has attracted the attention of researchers in computer languages for many years. For visual

* Corresponding author.
E-mail address: mulhem@kfupm.edu.sa (M. Al-Mulhem).

languages to be accepted as an alternative for textual languages there is a need for a lot more work in the theoretical background of this area. Recently there are various attempts in that direction for developing grammatical formalism, parsing algorithms, and compiler generator tools for visual languages.

Syntactic, interactive, learning, icon-oriented system compiler (SILICON Compiler) [20] is a software system for the specification, interpretation, prototyping and generation of icon-oriented systems. It is based on the concept of generalized icon. A generalized icon is represented as (X_m, X_i) where X_m is called the logical part (semantics), and X_i is called the physical part (syntax). It uses picture grammar to specify the physical part of the icons (concrete syntax of a visual language). A picture grammar is a context-free grammar where terminal symbols include both primitive picture elements and spatial operators.

Visual grammars [14] are used for describing VPL. A *visual grammar* is basically a context free grammar annotated to indicate the spatial arrangement of picture elements. The right-hand side (RHS) of each production is a spatial template indicating the picture components and spatial arrangement among them. The RHS elements can be visual literals (i.e., terminals) or nonterminals. In this formalism an input sentence is viewed as a spatial arrangement of text-graphic symbols where each text-graphic symbol is either a fragment of text or a primitive graphical element. A top down spatial parser based on this formalism has been developed with exponential time complexity [12].

Positional grammars are used as a grammatical formalism for the specification and parsing of VPL [9,10]. It is an extended version of the context-free grammar. A generalized LR parsing algorithm [1] called DR parser [9] has been developed for this formalism. The differences between LR parser and DR parser are mainly in the parsing table. The DR parsing table has an additional field called position, which specify the position of the next symbol to be accessed in a particular state (state on top of the stack). Working of the DR parser is similar to the LR parser except that it accesses the next symbol at the position indicated in the position field of the state on top of the stack. The generation method for the DR parse table is similar to the LR method except for few modifications. The disadvantage of this formalism is that it may not parse languages generated by ambiguous positional grammars.

Picture layout grammars (PLGs) [12] is basically a restricted form of attributed multiset grammar. In PLGs a visual sentence is represented as a multiset of attributed symbols. The attributes of a symbol denote its location, size, color and so on. An inefficient parsing algorithm for PLGs has been developed. The parser was applied to actual language (Statecharts) and the performance was found to be bounded by $O(n^3)$, where n is number of symbols in the input sentence.

The family of grammatical formalisms, which represent visual sentences as multiset of icons with certain relationships among them, are loosely termed as *relational grammars* (RGs) [23]. The languages generated by relational grammars are called relational languages. A sentence in a relational language is represented as a multiset of symbols and relations. A bottom up parsing algorithm for unrestricted class of relational grammars has been developed [13]. This algorithm has the advantage that input objects (icons of a visual sentence) can be scanned and composed in any order. The disadvantage of the algorithm is the lack of efficiency. To describe an efficient parsing algorithm a subclass of relational grammars called fringe relational grammars (FRGs) has been proposed [23] along with a recognition algorithm. FRGs describe visual sentences as partially ordered multiset of symbols and relations.

Relation grammars [3,7,17,18] are introduced as extensions of the textual grammars. In this formalism, information about replacement mechanism of productions and derivation process is made explicit. Production rules specify a set of symbols and the spatial relationships among those symbols. An inefficient parsing algorithm based on RG formalism has been developed [7,18]. It is a bottom-up parsing algorithm and it consists of two phases. In the first phase the input sentence is checked to see whether the input symbols belongs to the set of terminal symbols or not, and whether the relation instances defined on them are valid or not. A relation instance is said to be valid if the relation it uses belongs to the set of relations among the symbols. In the second phase it is verified whether the input sentence can be derived by means of production rules whose constraints are valid.

Relation grammar/1 (RG/1) [18] is defined by imposing restrictions both on the form of productions and on the evaluation rules. Though RG/1 formalism is not as expressive as RG, it is capable of describing visual languages of practical use. A top–down parsing algorithm based on RG/1 has been proposed [18]. It takes as input an RG/1 grammar and a visual sentence represented as (T, RT), where T is a set of symbols and RT is a set of relations. It produces a top–down parse for the sentence if it is grammatically correct. The time complexity of the above algorithm as derived in [7] is $O(n \log n)$, where n is the number of elementary icons in the input visual sentence.

One nonterminal S-item relation grammar (1NS-RG) [17] is another restricted form of RGs. It is an extension of Earley's algorithm [5]. In this formalism any constraint whether it appears on the RHS of a production or in an evaluation rule has at the most one nonterminal. A predictive parsing algorithm has been developed for this formalism. This algorithm is essentially a recognizer of visual sentences and it does not generate parse tree for the input sentence. It checks whether the input visual sentence belongs to $L(G)$, for a particular 1NS-RG grammar 'G', or not. This parsing algorithm, in general, has exponential time complexity. But when applied to languages having additional properties of connection and degree-boundedness, such as diagrammatic languages, it has polynomial time complexity.

The visual language compiler–compiler (VLCC) [8,11] is a grammar-based graphical system for the automatic generation of visual programming enviroments. Its grammatical formalism is based on 'positional grammar' and its parsing algorithm is an extension of the LR parser called pLR parser.

The symbol-relation grammar (SR) [6] represents visual sentences as a set of symbol occurrences and a set of relation items over symbol occurrences. The derivation process uses a rewriting rules that is based on context-free productions. The SR methodology have been exploited to construct parsers for visual languages [7,17,18].

This paper presents a restricted form of RG grammar called MRG grammar. The proposed grammar is based on RG/1 formalism [8,11,18]. It is obtained by restricting the form of productions, and distributing the evaluation rules among various productions rules. Section 2 gives the formal definition of the MRG grammar and it shows a number of examples to demonstrate the suitability and practicality of the formalism for expressing visual languages. This section also gives the definition for the languages defined by MRG and discusses the expressive power of the formalism. Section 3 presents an $O(n)$ parsing algorithm for the proposed formalism. This section also shows the complete parsing steps for a visual sentence and discusses the parser correctness and the time complexity of the parsing algorithm. Section 4 compares MRG with RG/1 formalism. Finally the paper concludes with final remarks.

2. Modified relation grammar

This section proposes a new grammatical formalism called *Modified Relation Grammar* (MRG) [2]. It is a restricted form of the relation grammar and is based on RG/1 formalism. It is obtained by restricting the form of productions, and distributing the evaluation rules among various productions rules. Definition 1 gives the formal definition of MRG.

Definition 1. A MRG is defined as a 5-tuple $G = (V_N, V_T, V_R, S, P)$, where

- V_N is a finite set of nonterminal symbols.
- V_T is a finite set of terminal symbols.
- V_R is a finite set of spatial relation symbols, $V_R = \{r_i \mid 1 \leqslant i \leqslant m\}$.
- S is the starting symbol.
- P is a finite set of productions, where each production is one of the following two types:
  1. $A := \{y, Y_1^{l_1}, Y_2^{l_2}, \ldots, Y_m^{l_m}\}\{r_1(y, Y_1^{l_1}),\ r_2(y, Y_2^{l_2}), \ldots, r_m(y, Y_m^{l_m}), R_1, R_2, \ldots, R_k\}\{E_1, E_2, \ldots, E_n\}$, where $y \in V_T, A \in V_N, Y_i^{l_i} \in V_N$ for $i = 1, \ldots, m$. The superscripts $l_1, l_2, \ldots, l_m$ on Y_is denotes the occurrence of the symbol on RHS of the production. If a nonterminal symbol $Y_i^{l_i}$ has only one occurrence on RHS, then it is superscripted as one. If there are two or more occurrences of a symbol $Y_i^{l_i}$, then different occurrences are superscripted as $1, 2, 3, \ldots$ and so on. $r_i \in V_R$ and for any $i, j \in \{1, \ldots, m\} r_i$

and r_j need not be distinct. $R_1, R_2, \ldots, R_k$ denotes possible additional constraints over the symbols $Y_i^{l_i}, i = 1, \ldots, m$. $E_1, E_2, \ldots, E_n$ denotes the external constraints of the production. These are nothing but the evaluation rules associated with the production. They specify how the symbols $\{y, Y_1^{l_1}, Y_2^{l_2}, \ldots, Y_m^{l_m}\}$ should be related to neighbors of A, when A is replaced by $\{y, Y_1^{l_1}, Y_2^{l_2}, \ldots, Y_m^{l_m}\}$ during a derivation step.

2. $A := \{y\}\{\}\{E_1, E_2, \ldots, E_n\}$, where $A \in V_N, y \in V_T, E_1, E_2, \ldots, E_n$ denotes the external constraints of the production.

Type 1 production states that the nonterminal symbol A, on input y, is to be replaced by the set $\{y, Y_1^{l_1}, Y_2^{l_2}, \ldots, Y_m^{l_m}\}$, provided the constraints $\{r_1(y, Y_1^{l_1}), r_2(y, Y_2^{l_2}), \ldots, r_m(y, Y_m^{l_m}), R_1, R_2, \ldots, R_k\}$ are satisfied. The first symbol on the RHS of a production (y) is called the primary symbol. The set of constraints $\{r_1(y, Y_1^{l_1}), r_2(y, Y_2^{l_2}), \ldots, r_m(y, Y_m^{l_m}), R_1, R_2, \ldots, R_k\}$ are called internal constraints. In internal constraints, the constraints $r_1(y, Y_1^{l_1}), r_2(y, Y_2^{l_2}), \ldots, r_m(y, Y_m^{l_m})$ are compulsory. The constraints $R_1, R_2, \ldots, R_k$ are optional, they denote possible additional constraints among the symbols $Y_1^{l_1}, Y_2^{l_2}, \ldots, Y_m^{l_m}$. If a production of type 1 is applied, then the symbol on the LHS will be replaced by the set of symbols on the RHS of the production, i.e., A will be replaced by the set $(y, Y_1^{l_1}, Y_2^{l_2}, \ldots, Y_m^{l_m})$. These symbols will be arranged according to the internal constraints. The set $\{E_1, E_2, \ldots, E_n\}$ denotes another set of constraints called the external constraints. They specify how the symbols of the RHS of a production are to be connected to the neighbors of the symbol on the LHS of the production, in a derivation tree, when the production is applied. An external constraints has the form:

$$t_j(X_j, A) : -[\text{constr}],$$

where $X_j \in y \cup \{Y_1, Y_2, \ldots, Y_m\}$, $t_j \in V_R$, for $j = 1, \ldots, n$.

The relation instance $t_j(X_j, A)$ is an internal constraint of some MRG production. [constr] is a set of constraints that specify how the constraint $t_j(X_j, A)$ will be reduced when the current production is applied. By reduction we mean how the constraint $t_j(X_j, A)$ is going to be changed when the current production is applied. These constraints are associated with the corresponding 'A' production as external constraints.

Type 2 production states that the nonterminal A, on input 'y', is to be replaced by 'y'. There are no internal constraints. The external constraints $E_1, E_2, \ldots, E_n$ has the same meaning as in type 1 production. 'y' in this case also, is the primary symbol.

One characteristic of MRG productions is that no two productions having the same LHS should have the same primary symbol. The following two examples should clarify the definition.

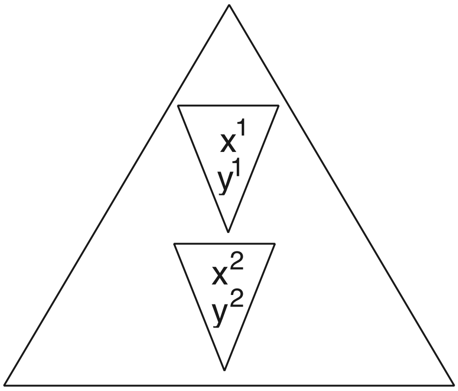

Fig. 1. A visual sentence.

Example 1. Let us give MRG description of the visual sentence shown in Fig. 1. MRG description for that sentence can be written as follows:

$$G := (V_N, V_T, V_R, S, P) := (\{S, A, B\}, \{\triangle, \bigtriangledown, x, y\}, \{\text{enclose}, \text{above}\}, S, P),$$

where P consists of:

1. $S := \{\triangle, A^1, A^2\}$ $\{\text{enclose}(\triangle, A^1)$, $\text{enclose}(\triangle, A^2)$, $\text{above}(A^1, A^2)\}$;
2. $A := \{\bigtriangledown, B^1, B^2\}$ $\{\text{enclose}(\bigtriangledown, B^1)$, $\text{enclose}(\bigtriangledown, B^2)$, $\text{above}(B^1, B^2)\}$ $\{\text{enclose}(\triangle^m, A)$:- $\text{enclose}(\triangle^m, \bigtriangledown)$; $\text{above}(A^m, A)$:- $\text{above}(A^m, \bigtriangledown)$; $\text{above}(A, A^m)$:- $\text{above}(\bigtriangledown, A^m)$; $\text{above}(\bigtriangledown^m, A)$:- $\text{above}(\bigtriangledown^m, \bigtriangledown)$; $\text{above}(A, \bigtriangledown^m)$:- $\text{above}(\bigtriangledown, \bigtriangledown^m)\}$;
3. $B := \{x\}$ $\{\}$ $\{\text{enclose}(\bigtriangledown^m, B)$:- $\text{enclose}(\bigtriangledown^m, x)$; $\text{above}(B^m, B)$:- $\text{above}(B^m, x)$; $\text{above}(B, B^m)$:- $\text{above}(x, B^m)\}$;
4. $B := \{y\}$ $\{\}$ $\{\text{enclose}(\bigtriangledown^m, B)$:-$\text{enclose}(\bigtriangledown^m, y)$; $\text{above}(B^m, B)$:- $\text{above}(B^m, y)$; $\text{above}(B, B^m)$:- $\text{above}(y, B^m)$; $\text{above}(x^m, B)$:- $\text{above}(x^m, y)\}$.

Example 2. Consider a visual language defined in [7]. A sentence in this language consists of 2D grids of two kinds of symbols. Symbols of type $\spadesuit$ are placed on the South-east frontier and the remaining grid positions are filled by $\clubsuit$ symbols. For example, v is a sentence in this language.

$$v = \begin{array}{l} \clubsuit^1\clubsuit^2\spadesuit^2 \\ \spadesuit^1\spadesuit^3\spadesuit^4 \end{array}$$

The MRG description for this sentence is written as follows:

$$G := (V_N, V_T, V_R, S, P) := (\{S, B, A\}, \{\clubsuit, \spadesuit\}, \{x, y, z\}, S, P),$$

where P consists of:

1. $S := \{\clubsuit, B^1, A^1\}$ $\{x(\clubsuit, B^1)$, $y(\clubsuit, A^1)$, $z(B^1, A^1)\}$;

2. $B := \{\clubsuit, B^1, A^1\}$ $\{x(\clubsuit, B^1),\ y(\clubsuit, A^1),\ z(B^1, A^1)\}$ $\{x(\clubsuit^m, B)$:- $x(\clubsuit^m, \clubsuit)$; $z(B, A^m)$:- $z(\clubsuit, A^m), x(A^m, A^1)\}$;
3. $B := \{\spadesuit, A^1\}$ $\{y(\spadesuit, A^1)\}$ $\{x(\clubsuit^m, B)$:- $x(\clubsuit^m, \spadesuit)$; $z(B, A^m)$:- $z(\spadesuit, A^m), x(A^m, A^1)\}$;
4. $A := \{\clubsuit, A^1\}$ $\{y(\clubsuit, A^1)\}$ $\{y(\clubsuit^m, A)$:- $y(\clubsuit^m, \clubsuit)$; $z(A, A^m)$:- $z(\clubsuit, A^m), x(A^m, A^1)$; $z(\spadesuit^m, A)$:- $z(\spadesuit^m, \clubsuit)$; $z(\clubsuit^m, A)$:- $z(\clubsuit^m, \clubsuit)$; $x(A, A^m)$:- $x(\clubsuit, A^m), z(A^m, A^1)$; $x(\clubsuit^m, A)$:- $x(\clubsuit^m, \clubsuit)\}$;
5. $A := \{\spadesuit\}$ $\{\}$ $\{y(\spadesuit^m, A)$:- $y(\spadesuit^m, \spadesuit)$; $y(\clubsuit^m, A)$:- $y(\clubsuit^m, \spadesuit)$; $z(\spadesuit^m, A)$:- $z(\spadesuit^m, \spadesuit)$; $z(\clubsuit^m, A)$:- $z(\clubsuit^m, \spadesuit)$; $z(A, A^m)$:- $z(\spadesuit, A^m)$; $x(A, A^m)$:- $x(\spadesuit, A^m)$; $x(\spadesuit^m, A)$:- $x(\spadesuit^m, \spadesuit)$; $x(A^m, A)$:- $x(A^m, \spadesuit)\}$.

Superscript 'm' on the symbols in the external constraints of the preceding two examples denote an arbitrary occurrence of the symbol.

A visual sentence in MRG is represented as a pair (T, RT), where T is a set of terminal symbols (icons) and RT is a set of spatial relations (constraints) among those symbols. The symbols in T and consequently the constraints in RT are totally ordered. They are ordered according to the ordering of the spatial relations in the set V_{R}. Let $r_1, r_2, \ldots, r_n$ be the spatial relations of the set V_{R}, with ordering $r_1 < r_2 < \cdots < r_n$. Then the symbols of T are ordered as follows:

Initially T is empty. First we start at an initial position, which is the minimum position of all relations $r_1, r_2, \ldots, r_n$. We add the symbol at this position to T. Next we add all the symbols among the immediate neighbors of this symbol which are linked to it via r_1 to T. Then we add all the symbols among its immediate neighbors which are linked to it via r_2 to T, and so on. Once all the symbols linked to the first symbol in T via the r_n relation are added, we go to the second symbol in T. For the second symbol in T we add all the symbols among its immediate neighbors which are linked to it via r_1 to T. These symbols will be added to T provided that they are not already included in T. Next we find all the symbols among immediate neighbors of the second symbol in T which are linked to it via r_2. We add these symbols to T, if they are not in T. We repeat this procedure for the remaining symbols in T until all the symbols of the input sentence are added to the set T.

The constraints part of the input sentence, i.e., the set RT is ordered as follows: First we add all the constraints involving the first symbol in T and its immediate neighbors to RT. Then we add all the constraints involving the second symbol in T and its immediate neighbors to RT. Only those constraints that are not already in RT are added at this step. Then we add all the constraints involving the third symbol in T and it's immediate neighbors, and so on. This procedure is repeated until all the constraints of the input sentence are added to the set RT. In this ordering we implicitly assumed that a constraint is specified between two immediately adjacent icons. The following two examples should clarify the ordering of the input sentences.

Example 3. Consider the visual sentence in Example 1. Assuming the set of relations V_R to be {enclose,above}, and ordering of relations to be enclose < above. The visual sentence in Fig. 1 can be ordered as

$$\begin{aligned} v &= (T, \mathrm{RT}) \\ &= (\{\triangle^1, \bigtriangledown^1, \bigtriangledown^2, x^1, y^1, x^2, y^2\}, \{\mathrm{enclose}(\triangle^1, \bigtriangledown^1), \mathrm{enclose}(\triangle^1, \bigtriangledown^2), \\ &\quad \mathrm{enclose}(\bigtriangledown^1, x^1), \mathrm{enclose}(\bigtriangledown^1, y^1), \mathrm{above}(\bigtriangledown^1, \bigtriangledown^2), \mathrm{enclose}(\bigtriangledown^2, x^2), \\ &\quad \mathrm{enclose}(\bigtriangledown^2, y^2), \mathrm{above}(x^1, y^1), \mathrm{above}(x^2, y^2)\}). \end{aligned}$$

Example 4. Consider the visual sentence in Example 2. Assuming the set of relations to be $V_R = \{x, y, z\}$, and $x < y < z$ to be their ordering. The visual sentence v

$$\begin{aligned} v = {} & \clubsuit^1 \clubsuit^2 \spadesuit^2 \\ & \spadesuit^1 \spadesuit^3 \spadesuit^4 \end{aligned}$$

can be ordered as

$$\begin{aligned} v &= (T, \mathrm{RT}) \\ &= (\{\clubsuit^1, \clubsuit^2, \spadesuit^1, \spadesuit^2, \spadesuit^3, \spadesuit^4\}\{x(\clubsuit^1, \clubsuit^2), y(\clubsuit^1, \spadesuit^1), x(\clubsuit^2, \spadesuit^2), y(\clubsuit^2, \spadesuit^3), \\ &\quad z(\clubsuit^2, \spadesuit^1), x(\spadesuit^1, \spadesuit^3), y(\spadesuit^2, \spadesuit^4), z(\spadesuit^2, \spadesuit^3), x(\spadesuit^3, \spadesuit^4)\}). \end{aligned}$$

2.1. Languages defined by MRG

This section gives the definition of the language generated by MRG grammar $G = (V_N, V_T, V_R, S, P)$. First the sentential form of G is defined, then the language generated by G is given.

Definition 2. A pair (T^1, RT^1), where T^1 is a set of symbols from $V_N \cup V_T$, and RT^1 is a set of spatial relations in V_R among symbols of T^1, is said to be a sentential form of MRG, if and only if there exists a derivation sequence

$$(\{S\}, \emptyset) \Rightarrow (T_1^1, \mathrm{RT}_1^1) \Rightarrow (T_2^1, \mathrm{RT}_2^1) \Rightarrow \cdots \Rightarrow (T_n^1, \mathrm{RT}_n^1) \Rightarrow (T^1, \mathrm{RT}^1).$$

In the above derivation sequence all the intermediate sentential forms are totally ordered, and the leftmost nonterminal is expanded at each step. In other words, the sentential form (T^1, RT^1) is obtained from $(\{S\},\emptyset)$ by constructing the derivation tree in a top down breadth first manner. Only these types of derivations will be permitted in MRG grammars. This is because, if a production is applied then all of its constraints have to be satisfied. All the constraints of a production can be satisfied only when all of its children are expanded in a breadth first manner.

Definition 3. Given the MRG Grammar $G = (V_N, V_T, V_R, S, P)$, the language defined by G, denoted by $L(G)$, is the set of all sentential forms (T, RT), where T consists of terminals only. In other words,

$$L(G) = \{(T, \mathrm{RT}) \\ |(\{S\}, \emptyset) \overset{*}{\Rightarrow} (T, \mathrm{RT}) \text{and } T \text{ consists of terminal symbols only}\}.$$

2.2. *Expressiveness of MRG*

This section shows that MRG grammar and RG/1 grammar are equivalent in terms of expressiveness power. To prove this, first we will give a procedure that will convert any RG/1 grammar to MRG grammar. This establishes the fact that MRG grammar is at least as expressive as RG/1 grammar. Then we will prove that the language accepted by RG/1 grammar is also accepted by the corresponding MRG grammar (MRG grammar obtained from RG/1 grammar by the given procedure).

Lemma 1. *MRG grammar is at least as expressive as RG/*1 *grammar.*

Proof. To prove this we will give a procedure called 'CONVERT' that will convert a given RG/1 grammar to MRG grammar. The input to the procedure is RG/1 grammar $G = (V_N, V_T, V_R, S, P, R)$ and the output is MRG grammar $G^1 = (V_N^1, V_T^1, V_R^1, S^1, P^1)$. The procedure is given below:

```
**Procedure** CONVERT($G, G^1$)
**Begin** {of procedure}
$V_T^1 := V_T$.
$V_R^1 := V_R$.
$S^1 := S$.
For each RG/1 production $p \in P$ of the form
$A := \{y, Y_1, Y_2, \ldots, Y_m\}\{r_1(y, Y_1), r_2(y, Y_2), \ldots, r_m(y, Y_m), R_1, R_2, \ldots, R_k\}$
We construct the corresponding MRG production/productions as follows:
    •  If all $Y_i s\, i = 1, \ldots, m$ are nonterminals then construct an MRG
       production as $A := \{y, Y_1^{l_1}, Y_2^{l_2}, \ldots, Y_m^{l_m}\}$ $\{r_1(y, Y_1^{l_1}), r_2(y, Y_2^{l_2}), \ldots,$
       $r_m(y, Y_m^{l_m}), R_1, R_2, \ldots, R_k\}$ $\{E_1, E_2, \ldots, E_n\}$, where $l_1, l_2, \ldots, l_m$ de-
       note different occurrences of the symbols $Y_1, Y_2, \ldots, Y_m$ on RHS of
       production. Add this production to the set $P^1$. Add
       $A, Y_1, Y_2, \ldots, Y_m$ to the set $V_N^1$. The external constraints
       $E_1, E_2, \ldots, E_n$ are the evaluation rules of the corresponding RG/1
       grammar. The evaluation rules of the RG/1 grammar are of the
       form:
```

$t_1(X_1, A)$:- $(A \Rightarrow \{y, Y_1, Y_2, \ldots, Y_m\}), q_1(X_1, Z_1)$
$\vdots$
$t_n(X_n, A)$:- $(A \Rightarrow \{y, Y_1, Y_2, \ldots, Y_m\}), q_n(X_n, Z_n)$
where X_j and $Z_j \in y \cup \{Y_1, Y_2, \ldots, Y_m\}, t_j$ and $q_j \in V_R$, for $j = 1, \ldots, n$.
These evaluation rules are rewritten in MRG grammar as
$t_1(X_1, A)$:- $q_1(X_1, Z_1)$
$\vdots$
$t_n(X_n, A)$:- $q_n(X_n, Z_n)$
and associated with the corresponding MRG 'A' production as external constraints.

- If some Y_i, let us say $Y_k, k \in \{1, \ldots, m\}$, is a terminal then we create a nonterminal W_k and the following MRG production
$W_k := \{Y_k\}\ \{\}\ \{E_1, E_2, \ldots, E_n\}$
where $E_1, E_2, \ldots, E_n$ are the external constraints of the form:
$t_1(X_1, W_k)$:- $t_1(X_1, Y_k)$
$\vdots$
$t_n(X_1, W_k) : -t_n(X_1, Y_k)$
where $t_1(X_1, Y_k), \ldots, t_n(X_1, Y_k)$ are constraints involving Y_k in RG/1 productions. Add W_k to the set V_N^1, and the production $W_k := \{Y_k\}\{\}\{E_1, E_2, \ldots, E_n\}$ to the set P^1. We create another MRG production (corresponding to the given RG/1 production) of the form:
$A := \{y, Y_1^{l_1}, Y_2^{l_2}, \ldots, W_k^{l_k}, \ldots, Y_m^{l_m}\}\ \{r_1(y, Y_1^{l_1}), r_2(y, Y_2^{l_2}), \ldots,$
$r_k(y, W_k^{l_k}), \ldots, r_m(y, Y_m^{l_m}), R_1, R_2, \ldots, R_k\}\ \{E_1, E_2, \ldots, E_n\}$
where $l_1, l_2, \ldots, l_m$ denotes different occurrences of symbols $Y_1, Y_2, \ldots, Y_m$ on RHS of production. $E_1, E_2, \ldots, E_n$ are the external constraints of the MRG production. The external constraints in this case are nothing but the evaluation rules of the corresponding RG/1 grammar, which are rewritten as discussed earlier. Add this production to the set P^1. Finally, add $A, Y_1, Y_2, \ldots, Y_m$ (if they are already not added) to the set V_N^1.

End {of procedure}

Lemma 2. *If $L = L(G)$ for some RG/1 grammar G, then $L = L(G^1)$ for the corresponding MRG grammar G^1.*

Proof. We will prove this lemma in two parts. First we will prove that all $v \in L(G)$ also belongs to $L(G^1)$. Then we will prove that if a sentence $v \notin L(G)$, then $v \notin L(G^1)$.

Part 1: $L(G)$ for RG/1 grammar G is defined as $L(G) = \{(T, \text{RT}) \mid (S, \emptyset) \overset{*}{\Rightarrow} (T, \text{RT})$ and T consists of terminal symbols only$\}$.

In the derivation of any sentence in G, at any derivation step, only the leftmost nonterminal in the sentential form will be expanded, i.e., the sentence (T,RT) will be obtained by the following sequence of derivations:

$$(\{S\},\emptyset) \Rightarrow (T_1,\mathrm{RT}_1) \Rightarrow (T_2,\mathrm{RT}_2) \Rightarrow \cdots \Rightarrow (T_n,\mathrm{RT}_n) \Rightarrow (T,\mathrm{RT}),$$

where at each step the leftmost nonterminal in $T_i(1 \leqslant i \leqslant n)$ will be expanded. In other words the sentence (T,RT) will be derived from S by expanding the derivation tree in top–down breadth first order.

In the MRG grammar obtained by procedure 'CONVERT' of Lemma 1, $S^1 = S$ and all the leftmost nonterminals expanded at various steps in RG/1 (nonterminals in $T_i(1 \leqslant i \leqslant n)$) belong to V_N^1. Also, there exists some productions involving these nonterminals in G^1 which has the same RHS as G productions. Apart from this, the derivation in MRG also proceeds in top–down breath first manner. From all these facts it is clear that every sentence generated by RG/1 grammar G can also be generated by MRG grammar G^1.

Part 2: Now let us prove that if a sentence $v = (T,\mathrm{RT}) \notin L(G)$ then it does not belong to $L(G^1)$.

A sentence $v = (T,\mathrm{RT}) \notin L(G)$ if there does not exist a leftmost derivation for it. Such a case is possible when a leftmost nonterminal in a particular sentential form cannot be expanded. Since $V_\mathrm{N} \subseteq V_\mathrm{N}^1$, this nonterminal also belongs to V_N^1 . Because the derivation of the sentential forms in MRG and RG/1 grammars is the same and $S^1 = S$, such a nonterminal also appears in MRG sentential form; and it cannot be expanded. So MRG derivation for v is not possible. In other words $v \notin L(G^1)$.

3. The MRG parsing algorithm

The MRG parsing algorithm is a table based predictive parser. The parser takes as input a visual sentence represented as the pair (T,RT), where T is a set of symbols and RT is a set of constraints among these symbols. We assume that both T and RT are totally ordered as explained earler. An outline of the parser is given in the next section (Section 3.1) and the details of the parsing algorithm is given in the following one (Section 3.2).

3.1. The parser

The system diagram of the MRG parser is shown in Fig. 2. The visual sentence to be parsed is stored in an input buffer called INP which consist of two parts; symbols part and constraints part. The symbols part stores the symbols of the input sentence (the set of icons in T); and the constraints part stores the constraints of the input sentence (the set of constraints in RT). The parser uses the following data structures:

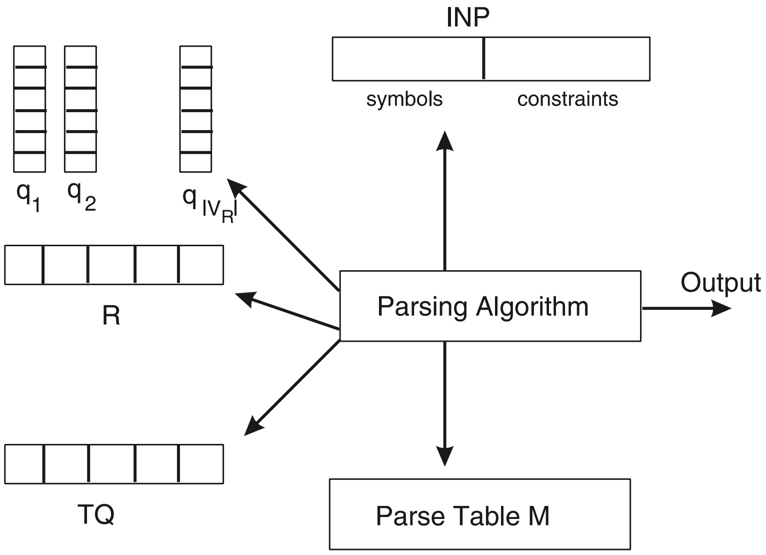

Fig. 2. The system diagram of the MRG parser.

1. A total of $|V_R|$ temporary queues $(q_1, q_2, \ldots, q_{|V_R|})$. These queues are created at the beginning of the parsing process to hold the intermediate nodes of the parse tree.
2. A data structure called R. This data structure is used to hold the internal constraints of the applied productions which are not yet resolved. These constraints come from the applied productions.
3. A data structure called TQ. This data structure contains all the constraints, involving all the symbols considered so far, at a particular parsing step. These constraints come from the input sentences.
4. A parse table M, which is organized as a two-dimensional array. For a given MRG grammar the nonterminals along the rows and terminals along the columns index the parse table M. For example a production of the form $A := \{y, Y_1^{l_1}, Y_2^{l_1}, \ldots, Y_m^{l_m}\}$ {internal constraints} {external constraints} will be stored in row indexed by 'A' and column indexed by 'y', in the parse table. An entry of the parse table could be a blank, or it may store a production of the grammar. Blank entries denote errors.

The parsing algorithm works top–down and explores the nodes of the parse tree in a breadth first manner. At a particular step during parsing, the algorithm tries to find a production $M(A, a)$, where A is the nonterminal in the parse tree to be expanded and 'a' is the current input symbol. If no production is found it reports an error and stops. If a production is found, the parser adds the internal constraints of the production to R. The parser then removes all the constraints involving current input symbol '*a*' from RT and adds them to TQ. Then the parser tries to resolve the constraints in R involving the current occurrence of A. If a constraint is fully resolved (i.e., if it contains only terminals), then it is removed from both R and TQ. Finally, the input symbol '*a*' is removed from T and the parsing process repeats with the next input symbol and the next parse tree node in the breadth first order.

In brief, the parsing algorithm produces the top–down parse of the input sentence as it traverses the parse tree in a breadth first manner. At a particular step during parsing it applies a production and adds the internal constraints of the production to R. These constraints will be resolved and satisfied at later steps of parsing.

3.2. The algorithm

This section presents the MRG parsing algorithm (Algorithm 1). Initially a total of $|V_R|$ queues $(q_1, q_2, \ldots, q_{|V_R|})$ are created. In addition to the data structures mentioned in the previous sections, Algorithm 1 uses the following operations and variables:

1. Head(q) is a function which returns the first element of the queue q.
2. Enqueue(q, S) is a function which adds the element S to the end of the queue q.
3. Dequeue(q) is a function which removes the first element from the queue q.
4. INP.symbols and INP.constraints return the symbols and constraints parts of the INP buffer. Both parts of the INP buffer are organized as queues.
5. The variables k, l, m, i and p are used in Algorithm 1, where k is a counter, l represent the occurrence of the nonterminals on the RHS of a production, m is the maximum number of symbols excluding the primary symbol on the RHS of a production, i is the level of the parse tree and p is a counter.

Algorithm 1. The MRG parsing algorithm.

```
Input: Totally ordered visual sentence $v$ and parsing table $M$ for MRG
grammar $G$.
Output: Top–down parse for $v$ if $v$ is in $L(G)$, otherwise error.
Method:
Initially, the input sentence $v$ in the form ($T$, RT) is stored in the input buffer
INP,
where $T$ is stored in the INP.symbols part and RT is stored in INP.con-
straints part.
Begin /* Method */
   Create $|V_R|$ queues $(q_1, q_2, \ldots, q_{|V_R|})$;
   Enqueue$(q_1, S^0)$;
   $k := 0; i := 1; p := 1; q\_no := 1; TQ := \{\}; R := \{\};$
```

```
While (INP.symbols not empty) do
Begin
  If (queue $q$_no not empty) then
    $A :=$ Head($q$_no);
    If (superscript on $A <= p$) then
      Dequeue($q$_no);
      $a :=$ Head(INP.symbols);
      If ($M(A, a)$ not blank) then
        Apply production $M(A, a)$;
        Print the pair $(A, a)$;
        Add internal constraints to $R$. Prior to adding the constraints
        to $R$, nonterminals in the internal constraints are attached
        superscript i and subscript $k + l$, and primary symbol is
        attached superscript of '$a$';
        Remove the constraints involving current input symbol '$a$'
        from INP.constraints and add them to TQ;
        Resolve the constraints in $R$ according to the external
        constraints. Those constraints that appear in both
        $R$ and TQ should be removed from both;
        Insert the symbol $X^i_{k+l}$ of the RHS of the applied production
        into the queue numbered 'pos' where 'pos' is the position
        of $r$ in $V_R$;
        If ($p! = i$) then
          Dequeue(INP.symbols);
          $k := k + m$;
        else
          $i := i + 1$;
          Dequeue(INP.symbols);
          $k := k + m$;
        endif;
      else /* else of 'If ($M(A, a)$ not blank)' */
        Print error message and stop;
      endif;
    else /* else of 'If (subscript on $A <= p$)' */
      $q$_no := $q$_no + 1;
      If ($q$_no <= $|V_R|$) then
        $k := k + m$;
      else
        $p := p + 1; i := i + 1;$ $q$_no := $q$_no + 1; $k := k + m$;
      endif;
    endif; /* endif of If (subscript on $A <= p$) */
  else /* else of 'If (queue $q$_no not empty' */
          If (All queues empty) then
             Print error message and stop;
          else
             $q\_no := q\_no + 1$;
             If ($q\_no <= |V_R|$) then
                $k := k + m$;
             else
                $p := p + 1; i := i + 1; q\_no := 1; k := k + m$;
             endif;
          endif;
       endif; /* endif of 'If (queue $q\_no$ not empty)' */
    end; /* end of main while loop */
    If ($q_1, q_2, \ldots, q_{|V_R|}$, $R$, TQ and INP.constraints are empty) then
       Print successful parsing and stop;
    else
       Print error message and stop;
    endif;
 end. /* method */
```

Example 5. Let us apply our parsing algorithm to the visual sentence of Example 1. The MRG grammar for this sentence is given in Example 1. It is also reproduced below for the sake of convenience.

$G := (V_N, V_T, V_R, S, P) := (\{S, A, B\}, \{\triangle, \triangledown, x, y\}, \{\text{enclose}, \text{above}\}, S, P)$ where P consists of

1. $S := \{\triangle, A^1, A^2\}\{\text{enclose}(\triangle, A^1), \text{enclose}(\triangle, A^2), \text{above}(A^1, A^2)\}$;
2. $A := \{\triangledown, B^1, B^2\}$ $\{\text{enclose}(\triangledown, B^1), \text{enclose}(\triangledown, B^2), \text{above}(B^1, B^2)\}$ $\{\text{enclose}(\triangle^m, A)$:- $\text{enclose}(\triangle^m, \triangledown)$; $\text{above}(A^m, A)$:- $\text{above}(A^m, \triangledown)$; $\text{above}(A, A^m)$:- $\text{above}(\triangledown, A^m)$; $\text{above}(\triangledown^m, A)$:- $\text{above}(\triangledown^m, \triangledown)$; $\text{above}(A, \triangledown^m)$:- $\text{above}(\triangledown, \triangledown^m)\}$;
3. $B := \{x\}$ $\{\}$ $\{\text{enclose}(\triangledown^m, B)$:- $\text{enclose}(\triangledown^m, x)$; $\text{above}(B^m, B)$:- $\text{above}(B^m, x)$; $\text{above}(B, B^m)$:- $\text{above}(x, B^m)\}$;
4. $B := \{y\}$ $\{\}$ $\{\text{enclose}(\triangledown^m, B)$:- $\text{enclose}(\triangledown^m, y)$; $\text{above}(B^m, B)$:- $\text{above}(B^m, y)$; $\text{above}(B, B^m)$:- $\text{above}(y, B^m)$; $\text{above}(x^m, B)$:- $\text{above}(x^m, y)\}$.

As shown in Example 3, this sentence has the following representation:

$$\begin{aligned} v &= (T, \text{RT}) \\ &= (\{\triangle^1, \triangledown^1, \triangledown^2, x^1, y^1, x^2, y^2\}, \{\text{enclose}(\triangle^1, \triangledown^1), \text{enclose}(\triangle^1, \triangledown^2), \\ &\quad \text{enclose}(\triangledown^1, x^1), \text{enclose}(\triangledown^1, y^1), \text{above}(\triangledown^1, \triangledown^2), \text{enclose}(\triangledown^2, x^2), \\ &\quad \text{enclose}(\triangledown^2, y^2), \text{above}(x^1, y^1), \text{above}(x^2, y^2)\}). \end{aligned}$$

The parsing steps for this sentence are as follows:

Initially: INP := ({INP.symbols},{INP.constraints}) := $(\{\triangle^1, \bigtriangledown^1, \bigtriangledown^2, x^1, y^1, x^2, y^2\}$, $\{$enclose$(\triangle^1, \bigtriangledown^1)$, enclose$(\triangle^1, \bigtriangledown^2)$, enclose$(\bigtriangledown^1, x^1)$, enclose$(\bigtriangledown^1, y^1)$, above$(\bigtriangledown^1, \bigtriangledown^2)$, enclose$(\bigtriangledown^2, x^2)$, enclose$(\bigtriangledown^2, y^2)$, above$(x^1, y^1)$, above$(x^2, y^2)\})$

$$q_1 := \{S^0\}; q_2 := \{\}; i := 1; p := 1; q\_\text{no} := 1; k := 0; \text{TQ} := \{\}; R := \{\};$$

Iteration 1: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing queue q_no, and assigning Head(INP.symbols) to 'a', we get

$$A := S^0; q_1 := \{\}; q_2 := \{\}; a := \triangle^1;$$

Production 1 (the content of $M(A, a)$) is applied. After adding the internal constraints to R we get

$$R := \{\text{enclose}(\triangle^1, A_1^1), \text{enclose}(\triangle^1, A_2^1), \text{above}(A_1^1, A_2^1)\}.$$

There are no constraints in R involving S^0, so no constraints are resolved.

After removing the constraints involving 'a' from INP.constraints and adding them to TQ we get

$$\text{TQ} := \{\text{enclose}(\triangle^1, \bigtriangledown^1), \text{enclose}(\triangle^1, \bigtriangledown^2)\}.$$

There are no constraints common to both R and TQ, so nothing is removed from R and TQ.

After inserting the symbols into the queues we get

$$q_1 := \{A_1^1, A_2^1\}; q_2 := \{\}; i := 2.$$

After dequeuing INP.symbols, we get

$$\begin{aligned}\text{INP} := (\{&\bigtriangledown^1, \bigtriangledown^2, x^1, y^1, x^2, y^2\}, \{\text{enclose}(\bigtriangledown^1, x^1), \text{enclose}(\bigtriangledown^1, y^1),\\ &\text{above}(\bigtriangledown^1, \bigtriangledown^2), \text{enclose}(\bigtriangledown^2, x^2), \text{enclose}(\bigtriangledown^2, y^2), \text{above}(x^1, y^1),\\ &\text{above}(x^2, y^2)\}), \quad i := 2; k := 2.\end{aligned}$$

Iteration 2: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing queue q_no, and assigning Head(INP.symbols) to 'a', we get

$$A := A_1^1; q_1 := \{A_2^1\}; q_2 := \{\}; a := \bigtriangledown^1.$$

Production 2 (the content of $M(A, a)$) is applied. After adding the internal constraints to R we get

$$\begin{aligned}R := \{&\text{enclose}(\triangle^1, A_1^1), \text{enclose}(\triangle^1, A_2^1), \text{above}(A_1^1, A_2^1), \text{enclose}(\bigtriangledown^1, B_3^2),\\ &\text{enclose}(\bigtriangledown^1, B_4^2), \text{above}(B_3^2, B_4^2)\}.\end{aligned}$$

After resolving the constraints in R we get

$$R := \{\text{enclose}(\triangle^1, \triangledown^1), \text{enclose}(\triangle^1, A_2^1), \text{above}(\triangledown^1, A_2^1), \text{enclose}(\triangledown^1, B_3^2), \text{enclose}(\triangledown^1, B_4^2), \text{above}(B_3^2, B_4^2)\}.$$

After removing the constraints involving 'a' from INP.constraints and adding them to TQ we get

$$\text{TQ} := \{\{\text{enclose}(\triangle^1, \triangledown^1), \text{enclose}(\triangle^1, \triangledown^2), \text{enclose}(\triangledown^1, x^1), \text{enclose}(\triangledown^1, y^1), \text{above}(\triangledown^1, \triangledown^2)\}.$$

The constraint enclose($\triangle^1, \triangledown^1$) is common to both R and TQ, so it is removed from both. R and TQ become

$$R := \{\text{enclose}(\triangle^1, A_2^1), \text{above}(\triangledown^1, A_2^1), \text{enclose}(\triangledown^1, B_3^2), \text{enclose}(\triangledown^1, B_4^2), \text{above}(B_3^2, B_4^2)\},$$

$$\text{TQ} := \{\text{enclose}(\triangle^1, \triangledown^2), \text{enclose}(\triangledown^1, x^1), \text{enclose}(\triangledown^1, y^1), \text{above}(\triangledown^1, \triangledown^2)\}.$$

After inserting the symbols into the queues we get

$$q_1 := \{A_2^1, B_3^2, B_4^2\}; q_2 := \{\}.$$

After dequeuing INP.symbols we get

$$\text{INP} := (\{\triangledown^2, x^1, y^1, x^2, y^2\}, \{\text{enclose}(\triangledown^2, x^2), \text{enclose}(\triangledown^2, y^2), \text{above}(x^1, y^1), \text{above}(x^2, y^2)\}), \quad k := 4.$$

Iteration 3: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing q_no, and assigning Head(INP.symbols) to 'a', we get

$$A := A_2^1; q_1 := \{B_3^2, B_4^2\}; q_2 := \{\}; a := \triangledown^2.$$

Production 2 (the content of $M(A, a)$) is applied. After adding the internal constraints to R we get

$$R := \{\text{enclose}(\triangle^1, A_2^1), \text{above}(\triangledown^1, A_2^1), \text{enclose}(\triangledown^1, B_3^2), \text{enclose}(\triangledown^1, B_4^2), \text{above}(B_3^2, B_4^2), \text{enclose}(\triangledown^2, B_5^2), \text{enclose}(\triangledown^2, B_6^2), \text{above}(B_5^2, B_6^2)\}.$$

After resolving the constraints in R we get

$$R := \{\text{enclose}(\triangle^1, \triangledown^2), \text{above}(\triangledown^1, \triangledown^2), \text{enclose}(\triangledown^1, B_3^2), \text{enclose}(\triangledown^1, B_4^2), \text{above}(B_3^2, B_4^2), \text{enclose}(\triangledown^2, B_5^2), \text{enclose}(\triangledown^2, B_6^2), \text{above}(B_5^2, B_6^2)\}.$$

After removing the constraints involving 'a' from INP.constraints and adding them to TQ, we get

$$\mathrm{TQ} := \{\mathrm{enclose}(\triangle^1, \bigtriangledown^2), \mathrm{enclose}(\bigtriangledown^1, x^1), \mathrm{enclose}(\bigtriangledown^1, y^1),$$
$$\mathrm{above}(\bigtriangledown^1, \bigtriangledown^2), \mathrm{enclose}(\bigtriangledown^2, x^2), \mathrm{enclose}(\bigtriangledown^2, y^2)\}.$$

The constraints enclose($\triangle^1, \bigtriangledown^2$) and above($\bigtriangledown^1, \bigtriangledown^2$) are removed from both R and TQ, so R and TQ become

$$R := \{\mathrm{enclose}(\bigtriangledown^1, B_3^2), \mathrm{enclose}(\bigtriangledown^1, B_4^2), \mathrm{above}(B_3^2, B_4^2), \mathrm{enclose}(\bigtriangledown^2, B_5^2),$$
$$\mathrm{enclose}(\bigtriangledown^2, B_6^2), \mathrm{above}(B_5^2, B_6^2)\},$$

$$\mathrm{TQ} := \{\mathrm{enclose}(\bigtriangledown^1, x^1), \mathrm{enclose}(\bigtriangledown^1, y^1), \mathrm{enclose}(\bigtriangledown^2, x^2), \mathrm{enclose}(\bigtriangledown^2, y^2)\}.$$

After inserting the symbols into the queues we get

$$q_1 := \{B_3^2, B_4^2, B_5^2, B_6^2\}; q_2 := \{\}.$$

After dequeuing INP.symbols we get

$$\mathrm{INP} := (\{x^1, y^1, x^2, y^2\}, \{\mathrm{above}(x^1, y^1), \mathrm{above}(x^2, y^2)\}), \quad k := 6.$$

Iteration 4: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, we see that the superscript on A is not less than or equal to p. So we increment the q_no, which now becomes 2; $k = 8$.

Iteration 5: In this iteration we find queue numbered 2 to be empty. We increment q_no, which now exceeds $|V_R|$. So q_no := 1; $p := 2$; $i := 3$; $k := 10$.

Iteration 6: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing q_no, and assigning Head(INP.symbols) to 'a', we get

$$A := B_3^2; q_1 := \{B_4^2, B_5^2, B_6^2\}; q_2 := \{\}; a := x^1.$$

Production 3 (the content of $M(A, a)$) is applied. Nothing is added to R since there are no internal constraints in this production.

After resolving the constraints in R we get

$$R := \{\mathrm{enclose}(\bigtriangledown^1, x^1), \mathrm{enclose}(\bigtriangledown^1, B_4^2), \mathrm{above}(x^1, B_4^2), \mathrm{enclose}(\bigtriangledown^2, B_5^2),$$
$$\mathrm{enclose}(\bigtriangledown^2, B_6^2), \mathrm{above}(B_5^2, B_6^2)\}.$$

After removing the constraints involving 'a' from INP.constraints and adding them to TQ we get

$$\mathrm{TQ} := \{\mathrm{enclose}(\bigtriangledown^1, x^1), \mathrm{enclose}(\bigtriangledown^1, y^1), \mathrm{enclose}(\bigtriangledown^2, x^2),$$
$$\mathrm{enclose}(\bigtriangledown^2, y^2), \mathrm{above}(x^1, y^1)\}.$$

After removing the common constraint enclose($\bigtriangledown^1, x^1$) from R and TQ we get

$$R := \{\mathrm{enclose}(\bigtriangledown^1, B_4^2), \mathrm{above}(x^1, B_4^2), \mathrm{enclose}(\bigtriangledown^2, B_5^2),$$
$$\mathrm{enclose}(\bigtriangledown^2, B_6^2), \mathrm{above}(B_5^2, B_6^2)\},$$

$$\mathrm{TQ} := \{\mathrm{enclose}(\bigtriangledown^1, y^1), \mathrm{enclose}(\bigtriangledown^2, x^2), \mathrm{enclose}(\bigtriangledown^2, y^2), \mathrm{above}(x^1, y^1)\}.$$

There are no symbols in this production to be inserted into the queues. After dequeuing INP.symbols we get

$$\mathrm{INP} := (\{y^1, x^2, y^2\}, \{\mathrm{above}(x^2, y^2)\}), \quad k := 12.$$

Iteration 7: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing q_no, and assigning Head(INP.symbols) to 'a', we get

$$A := B_4^2; q_1 := \{B_5^2, B_6^2\}; q_2 := \{\}; a := y^1.$$

Production 4 (the content of $M(A, a)$) is applied. Nothing is added to R since there are no internal constraints in this production.

After resolving the constraints in R we get

$$R := \{\mathrm{enclose}(\bigtriangledown^1, y^1), \mathrm{above}(x^1, y^1), \mathrm{enclose}(\bigtriangledown^2, B_5^2), \\ \mathrm{enclose}(\bigtriangledown^2, B_6^2), \mathrm{above}(B_5^2, B_6^2)\}.$$

Nothing is added to TQ at this step. Therefore,

$$\mathrm{TQ} := \{\mathrm{enclose}(\bigtriangledown^1, y^1), \mathrm{enclose}(\bigtriangledown^2, x^2), \mathrm{enclose}(\bigtriangledown^2, y^2), \mathrm{above}(x^1, y^1)\}.$$

After removing the common constraints enclose($\bigtriangledown^1, y^1$) and above (x^1, y^1) from R and TQ we get

$$R := \{\mathrm{enclose}(\bigtriangledown^2, B_5^2), \mathrm{enclose}(\bigtriangledown^2, B_6^2), \mathrm{above}(B_5^2, B_6^2)\},$$

$$\mathrm{TQ} := \{\mathrm{enclose}(\bigtriangledown^2, x^2), \mathrm{enclose}(\bigtriangledown^2, y^2)\}.$$

There are no symbols in this production to be inserted into the queues. After dequeuing INP.symbols we get

$$\mathrm{INP} := (\{x^2, y^2\}, \{\mathrm{above}(x^2, y^2)\}), \quad k := 14.$$

Iteration 8: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing q_no, and assigning Head(INP.symbols) to 'a', we get

$$A := B_5^2; q_1 := \{B_6^2\}; q_2 := \{\}; a := x^2.$$

Production 3 (the content of $M(A, a)$) is applied. Nothing is added to R since there are no internal constraints in this production.

After resolving the constraints in R we get

$$R := \{\mathrm{enclose}(\bigtriangledown^2, x^2), \mathrm{enclose}(\bigtriangledown^2, B_6^2), \mathrm{above}(x^2, B_6^2)\}.$$

Removing the constraint involving x^2 from INP.constraints and adding it to TQ we get

$$\mathrm{TQ} := \{\mathrm{enclose}(\bigtriangledown^2, x^2), \mathrm{enclose}(\bigtriangledown^2, y^2), \mathrm{above}(x^2, y^2)\}.$$

After removing the common constraint enclose($\bigtriangledown^2, x^2$) from R and TQ we get

$$R := \{\mathrm{enclose}(\bigtriangledown^2, B_6^2), \mathrm{above}(x^2, B_6^2)\},$$

$$\mathrm{TQ} := \{\mathrm{enclose}(\bigtriangledown^2, y^2), \mathrm{above}(x^2, y^2)\}.$$

There are no symbols in this production to be inserted into the queues. After dequeuing INP.symbols we get

$$\mathrm{INP} := (\{y^2\}, \{\}), \quad k := 16.$$

Iteration 9: After checking that the queue *q*_no is not empty, assigning the Head(*q*_no) to *A*, dequeuing *q*_no, and assigning Head(INP.symbols) to '*a*', we get

$$A := B_6^2; q_1 := \{\}; q_2 := \{\}; a := y^2.$$

Production 4 (the content of $M(A, a)$) is applied. Nothing is added to R since there are no internal constraints in this production.

After resolving the constraints in R we get

$$R := \{\mathrm{enclose}(\bigtriangledown^2, y^2), \mathrm{above}(x^2, y^2)\}.$$

No constraints are added to TQ because INP.constraints is empty. Therefore,

$$\mathrm{TQ} := \{\mathrm{enclose}(\bigtriangledown^2, y^2), \mathrm{above}(x^2, y^2)\}.$$

After removing the common constraints enclose($\bigtriangledown^2, y^2$) and above(x^2, y^2) from R and TQ we get

$R := \{\}$,

TQ $:= \{\}$.

There are no symbols in this production to be inserted into the queues.

After dequeuing INP.symbols we get

INP $:= (\{\}, \{\})$, $k := 18$.

Iteration 10: In this iteration INP.symbols is empty. Checks are made to see whether all queues (R, TQ, INP.constraints) are empty or not. All these queues are empty and therefore the algorithm announces successful completion of parsing.

3.3. Parser correctness

This section proves the correctness of the MRG parsing algorithm. The proof is established using the following three steps:

1. we will prove that the parsing algorithm halts in a finite time (after finite number of steps);
2. we will prove that given MRG grammar and a syntactically incorrect input sentence, the parsing algorithm will reject it and print an error message;

3. we will prove that given MRG grammar and a syntactically correct input sentence, the parsing algorithm will produce a correct top–down parse for the input sentence.

   The following lemma gives the proof for step 1.

Lemma 3. *The parsing algorithm will halt in a finite number of steps.*

Proof. Looking at the MRG parsing algorithm (Algorithm 1), we see that the main loop of the parsing algorithm is repeated 'n' times, where n is the number of symbols of the input sentence. Within this loop each step takes a constant (finite) time. If an input sentence is syntactically correct then the loop is executed 'n' times. After the termination of the loop the parsing algorithm terminates successfully. If an input sentence is syntactically wrong then the loop is exited prematurely and the algorithm terminates with an error message. So in all the cases (i.e., whether the input sentence is correct or wrong) the parsing algorithm terminates after a finite number of steps.

Before establishing the proof for the other two steps let us give some definitions.

Definition 4. A visual sentence $v = (T, \mathrm{RT})$ is said to be syntactically correct with respect to MRG grammar G, if $v \in L(G)$. The definition of $L(G)$ is given in Section 2.1.

Definition 5. A visual sentence $v = (T, \mathrm{RT})$ is said to be syntactically incorrect with respect to MRG grammar G, if $v \notin L(G)$.

Lemma 4. *Given MRG grammar for a visual language and a syntactically incorrect sentence* $v = (T, \mathrm{RT})$ *as input, the parsing algorithm will reject it with an error message.*

Proof. We will consider various cases under which a sentence becomes incorrect and show how they are detected.

Case 1: When a user constructs a visual sentence using terminal symbols from V_T but do not combine them according to the syntax of the language. For example the following sentence:

$$\begin{array}{l} \spadesuit^1\clubsuit^1 \\ \spadesuit^1\spadesuit^1 \end{array}$$

does not belong to $L(G)$ for G given in Example 2, even though the symbols it uses belongs to V_T. Such errors will be caught by the parsing algorithm during parsing when it searches for the production $M(A, a)$. The entry $M(A, a)$ in such a case will be blank causing the parser to reject the input sentence and prints an error message. In this case the parser finds that $M(S, ♠)$ is empty so it terminates the parsing process and prints an error message.

Case 2: When a user constructs an incomplete visual sentence. Let us say a user constructs the following sentence

$♣^1$

Even though the above sentence uses terminals from V_T, it is not a valid sentence of $L(G)$, for G given in Example 2. The parsing algorithm will catch such errors when it tries to expand a nonterminal in the parse tree but finds no input corresponding to it in the input sentence. For the given sentence, at first step the parser applies production 1 of Example 2 to expand S. After this step when it tries to expand B it finds no input. At this point the parser prints an error and stops.

We think that all the syntactically incorrect sentences fall into one of the above categories.

Lemma 5. *Given MRG grammar and a syntactically correct sentence, the parsing algorithm will produce a correct top–down parse tree for the sentence.*

Proof. As it is evident by now, our parsing algorithm constructs the parse tree for an input sentence in a top–down breadth first manner. Since this is the only type of derivation allowed in MRG (see the description of MRG sentential form in Section 2.1 and the parsing steps in Example 5), the parse tree produced by our algorithm is correct.

Theorem 1. *Let G be MRG grammar and* $v = (T, \mathrm{RT})$ *be an input sentence. The parsing algorithm will produce a correct parse iff* $v \in L(G)$.

Proof. Lemmas 3–5.

3.4. Parser time complexity

Lemma 6. *The number of times the main while loop of the parsing algorithm is repeated is* $n^*|V_R|$ *or* $O(n)$.

Proof. Consider Algorithm 1. In the worst case symbols are added to only one queue, remaining queues are kept empty. Also only one symbol is added to the queue at a time. Let us assume that after application of the first production (first iteration of the while loop or first step of parsing), symbols are added only to the last queue ($q_{|V_R|}$). The initial configuration of the queues will be as follows:

$$q_1 := \{S^0\}; q_2 := \{\}; q_3 := \{\}, \ldots, q_{|V_R|} := \{\}.$$

In the first iteration S^0 will be expanded and a nonterminal symbol, let us say A (written as A_1^1), will be inserted into $q_{|V_R|}$. In the second iteration q_1 is found to be empty so the parser goes to q_2. Since q_2 is also empty, it goes to q_3, and so on until it reaches $q_{|V_R|}$. The number of iterations required to reach $q_{|V_R|}$ is $q_{|V_R|-1}$. Once $q_{|V_R|}$ is reached, the symbol A_1^1 will be expanded to another symbol, let us say B (written as B_s^2 where $s = |V_R|^* m + 1$). At the end of this iteration only two symbols of the input sentence are parsed. The first one is parsed when S is expanded, and the second one when A is expanded. The parsed symbols are primary symbols of applied productions. Still $(n-2)$ symbols of the input sentence are to be parsed.

In the next iteration (the iteration immediately after the iteration in which A is expanded) the parser finds the superscript of B to be greater than p (p is initialized to 1), so it loops back and goes to q_1 after incrementing p and i. It finds q_1 to be empty, goes to q_2, and so on till it reaches $q_{|V_R|}$. At this point it expands B. So to expand every nonterminal on $q_{|V_R|}$ the parser performs $|V_R| + 1$ iterations. Since there are $(n-2)$ symbols yet to be analyzed, the parser needs $(n-2)^*(|V_R|+1)$ iterations. Therefore the total number of iterations required to parse a sentence of 'n' symbols is $I = 1$ iteration for expansion of $S +$ $(|V_R| - 1)$ iterations required to reach $q_{|V_R|} + 1$ iteration to expand $A + (n-2)^*$ $(|V_R| + 1)$ iterations required to parse the remaining $n-2$ symbols.

Therefore,

$$I = 1 + |V_R| - 1 + 1 + (n-2)^*(|V_R| + 1) = (n-1)^*(|V_R| + 1).$$

Assuming $n \gg 1, |V_R| \gg 1$ we get

$$I = n^*|V_R|.$$

So in the worst case the while loop is repeated $n^*|V_R|$ times. Since $|V_R|$ is constant for a given grammar, we can say that the number of times the main while loop repeated is $O(n)$.

Theorem 2. *The time complexity of the parsing algorithm is* $O(n)$.

Proof. Let us derive the time complexity for the parsing algorithm. The statement for creating $|V_R|$ queues takes $O(|V_R|)$ time. The statements for initializing q_1, k, i, p, q_no, TQ and R takes constant time.

The main while loop of the parsing algorithm is repeated at most $n^*|V_R|$ times(see Lemma 6). Within the while loop, the time taken by various statements is as follows:

The statement that checks whether INP.symbols is empty is executed in constant time. The statement which checks whether all queues are empty takes $O(|V_R|)$ time. The following blocks of statements also take constant time;
- Is queue numbered q_no empty.
- $A :=$ Head(q_no).

- Is superscript on $A \leqslant p$.
- Dequeue(q_no).
- a: = Head(INP.symbols).
- q_no := q_no + 1.
- Is q_no = $|V_\mathrm{R}| + 1$.
- $p := P + 1$.
- $i : i + 1$.
- q_no := 1.
- Is TQ empty.
- Is INP.constraints empty.

Since the parse table is organized as 2D array indexed by nonterminals along the rows and terminals along the columns, the statement 'Is $M(A, a)$ blank?' takes constant time. The time taken for applying a production $M(A, a)$ and adding its internal constraints to R is constant; because once the grammar is given we can consider the number of symbols and the internal constraints of any production as constants. Since the constraints of the input sentence are totally ordered, the constraints involving current input symbol 'a' appears towards the beginning of INP.constraints. The time taken to remove these constraints and add it to TQ, is constant. Since R is of constant length and can be organized as an array, constraints involving current occurrence of nonterminal A can be accessed in constant time. These constraints can then be resolved. A fully resolved constraint can be removed from both R and TQ. This operation also takes constant time. The time taken to insert symbols on the RHS of an applied production into queues is constant, because once the grammar is given the number of symbols on the RHS of any MRG production is constant. The time taken for the statements 'Is $p = i$' and 'Dequeue(INP.symbols)' is constant.

So time taken for 1 iteration of while loop is bounded by some constant, let us say δ_1. δ_1 could be $O(|V_\mathrm{R}|)$, or δ, where δ is time taken by any other statement in the while loop. Since the while loop is repeated $n^*|V_\mathrm{R}|$ times; the time taken for execution of the while loop is $n^*|V_\mathrm{R}|^*\delta_1$.

Total time complexity of the algorithm = Time taken for the execution of statements occurring before the while loop + Time taken for the execution of the while loop body.

The time taken by the execution of statements before the while loop is constant, therefore

Total time complexity of the algorithm
= Time taken for the execution of the while loop body = $n^*|V_\mathrm{R}|^*\delta_1$.

$|V_\mathrm{R}|$ and δ_1 are constant, therefore the time complexity of the algorithm is O(n).

4. Comparison of MRG with RG-based formalisms

This section presents in details a comparison between the MRG and RG/1 formalisms, because of the similarity between the two. It has been shown that MRG and RG/1 grammars have the same expressive power (Lemma 1). The main differences between MRG and RG/1 grammars are the following:

(1) In MRG grammar, the evaluation rules are associated with the productions as external constraints. In RG/1 grammar, the evaluation rules are stored in a separate set R. The association of evaluation rules with the productions, in MRG, helps in improving the clarity of productions. It is clear how the symbols on RHS of a production should be linked to the neighbors of the symbol on the LHS of a production, in a derivation tree, when the production is applied. In RG/1 formalism the evaluation rules are maintained in a separate set R and hence it is not clear how the symbols on RHS of a production should be linked to the neighbors of symbol on LHS of a particular production, in a derivation tree, when the production is applied. In MRG the evaluation rules are distributed among all the productions except the first production (production having starting symbol S as LHS). Due to the distribution of evaluation rules in MRG the time taken to apply a production during parsing is reduced.

(2) In MRG grammar we have two types of productions, whereas in RG/1 we have only one type of production. In type 1 production of MRG, the symbols on RHS other than primary symbol should be nonterminals. In RG/1 productions these symbols can be terminals or nonterminals. This factor however does not reduce the expressiveness of MRG because, for each terminal on RHS of an RG/1 production, other than primary symbol, we can create a nonterminal and use it in place of the terminal, in RHS of corresponding MRG production. Of course we have to add a type 2 production having the newly created nonterminal as LHS and the terminal symbol as RHS Example 6 will clarify this procedure.

Example 6. If we have an RG/1 production

$$A := \{a, Y_1, Y_2, b\}\{x(a, Y_1), y(a, Y_2), z(a, b)\},$$

where A, $Y_1, Y_2 \in V_N$ and $a, b \in V_T$.

The corresponding MRG production set can be written as

$$A := \{a, Y_1^1, Y_2^1, Y_3^1\}\{x(a, Y_1^1), y(a, Y_2^1), z(a, Y_3^1)\}\{\},$$

$$Y_3 := \{b\}\{\}\{z(a, Y_3) : -z(a, b)\},$$

where Y_3 is the newly created nonterminal.

It is clear from Example 6 that the above type of substitution increases the sizes of production and nonterminal sets. Since the sizes of nonterminal and production sets increase just by a constant amount, the asymptotic time complexity of the MRG parsing algorithm will not be affected.

(3) In type 1 production of MRG the primary symbol can be related to all other symbols on RHS by arbitrary relation symbols. These relation symbols need not be distinct. In RG/1 grammar the primary symbol is to be related to all the other symbols of RHS via distinct relation symbols. This restriction reduces the clarity of RG/1 grammars. Sentences such as the one shown in Fig. 1 cannot easily be expressed using RG/1 grammar.

Theorem 2 shows that the time complexity of the MRG parsing algorithm is $O(n)$, which is more efficient than RG/1 parsing algorithm with time complexity $O(n \log n)$. Both parsing algorithms work top–down and construct the parse tree in a breadth first order. Both of these algorithms use a main loop which is repeated n times, where '*n*' is the number of symbols in the input sentence. The difference between these two algorithms is in the way they resolve the internal constraints. In RG/1 parsing algorithm, when the algorithm finds that two symbols are potentially context identical it saves the information of all the *dpcis* (description of potentially contextual identities) in the set *D*. The time taken for constructing, inserting, and deleting *dpcis* is $O(\log n)$.

On the other hand, MRG parsing algorithm does not use the concept of *dpcis*. It uses superscript and subscripts on nonterminals to identify internal constraints involving them. The set of internal constraints that are not yet resolved can be organized as a 3D array indexed by nonterminals, superscripts, and subscripts. Since access time in an array is constant, the constraints involving a particular occurrence of a nonterminal can be accessed in constant time.

5. Conclusions

We have proposed a new grammatical formalism called MRG to model VPL. This is basically a restricted form of the RG obtained by restricting the type of productions. It has been shown that the expressive power of MRG grammar is equivalent to RG/1 grammar. However, MRG grammar is superior to RG/1 grammar in terms of clarity of grammar productions.

An efficient MRG parsing algorithm is also proposed. It has been shown that the MRG parsing algorithm is more efficient than the RG/1 parsing algorithm. The time complexity of the parsing algorithm is derived and is found to be linear. A number of examples of visual sentences have been used to clarify the MRG grammar and its parsing algorithm. The working of the parsing algorithm has been demonstrated by tracing the parsing of one visual sentence and showing in details the parsing steps.

Acknowledgements

We would like to thank KFUPM for providing the computing facilities to carry out this work.

References

[1] A. Aho, R. Sethi, J. Ullman, Compilers: Principles Techniques and Tools, Addison-Wesley, Reading, MA, 1986.

[2] M.A. Ahmed, A parsing algorithm for iconic visual programming languages, Master's thesis, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, December 1996.

[3] C. Crimi, A. Guerico, G. Nota, G. Pacini, G. Tortora, M. Tucci, Relation grammars for modelling multi-dimensional structures, in: Proceedings of the IEEE Workshop on Visual Languages, IEEE Computer Society Press, Silver spring, MD, 1990, pp. 168–173.

[4] S. Chang, Ten years of visual languages research, in: Proceedings of the IEEE Symposium on Visual Languages, IEEE Computer Society Press, Silver spring, MD, 1994, pp. 196–205.

[5] J. Earley, An efficient context-free parsing algorithm, Communications of the ACM 13 (2) (1970) 94–102.

[6] F. Ferrucci, G. Pacini, G. Satta, M.I. Sessa, G. Tortora, M. Tucci, G. Vitiello, Symbol-relation grammars: a formalism for graphical languages, Information and Computation 131 (0090) (1996) 1–46.

[7] F. Ferrucci, G. Pacini, G. Tortora, M. Tucci, G. Vitiello, Efficient parsing of multi-dimensional structures, in: Proceedings of the IEEE Workshop on Visual Languages, IEEE Computer Society Press, Silver spring, MD, 1991, pp. 105–110.

[8] G. Costagliola, A. DeLucia, S. Orefice, G. Tortora, A parsing methodology for the implementation of visual systems, IEEE Transactions on Software Engineering 23 (12) (1997) 777–799.

[9] G. Costagliola, S. Chang, DR PARSERS: A generalization of LR parsers, in: Proceedings of the IEEE Workshop on Visual Languages, IEEE Computer Society Press, Silver spring, MD, 1990, pp. 174–180.

[10] G. Costagliola, M. Tomita, S. Chang, A generalized parser for 2D languages, in: Proceedings of the IEEE Workshop on Visual Languages, IEEE Computer Society Press, Silver spring, MD, 1991, pp. 98–104.

[11] G. Costagliola, S. Orefice, A. DeLucisa, Automatic generation of visual programming enviroments, IEEE Computer (1995) 56–66.

[12] E. Golin, A method for the specification and parsing of visual languages, Ph.D. thesis, Brown University, USA, May 1991.

[13] K. Wittenburg, L. Weitzman, J. Talley, Unification-based grammars and tabular parsing for graphical languages, Journal of Visual Languages and computing 2 (1991) 347–370.

[14] F. Lakin, Spatial parsing for visual languages, in: S. Chang, T. Ichikawa, P. Ligomenides (Eds.), Visual Languages, Plenum Press, New York, 1989, pp. 35–85.

[15] M. Hirakawa, N. Monden, I. Yoshimoto, M. Tanaka, T. Ichikawa, HI-VISUAL: a language supporting visual interaction in programming, in: S. Chang, T. Ichikawa, P. Ligomenides (Eds.), Visual Languages, Plenum Press, New York, 1989, pp. 233–259.

[16] M. Pong, N. Ng, PIGS-a system for programming with interactive graphical support, Software – Practice and Experience 13 (9) (1983) 847–855.

[17] M. Tucci, F. Ferrucci, G. Tortora, G. Vitiello, A predictive parser for visual languages specified by relation grammars, in: Proceedings of the IEEE Workshop on Visual Languages, IEEE Computer Society Press, Silver Spring, MD, 1994, pp. 245–252.

[18] M. Tucci, G. Vitiello, G. Costagliola, Parsing nonlinear languages, IEEE Transactions on Software Engineering 20 (9) (1994) 720–739.
[19] R. Korfhage, M. Korfhage, Criteria for iconic languages, in: S. Chang, T. Ichikawa, P. Ligomenides (Eds.), Visual Languages, Plenum Press, New York, 1989, pp. 207–231.
[20] S. Chang, M. Tauber, B. Yu, J. Yu, A visual language compiler, IEEE Transactions on Software Engineering 15 (10) (1989) 506–525.
[21] S. Chang, T. Ichikawa, P. Ligomenides, Visual Languages, Plenum Press, New York, 1989.
[22] N. Shu, Visual Programming, Van Nostrand Reinhold, New York, 1988.
[23] K. Wittenburg, Earley-style parsing for relational grammars, in: Proceedings of the IEEE Workshop on Visual Languages, IEEE Computer Society Press, Silver Spring, MD, 1992, pp. 192–199.