RGG+: An Enhancement to the Reserved Graph Grammar Formalism

Xiaoqin Zeng[1,2] Kang Zhang[1] Jun Kong[1] Guang-Lei Song[1]

[1]Department of Computer Science, University of Texas at Dallas, Texas 75083, USA

[2]Department of Computer Science, Hohai University, Nanjing 210098, China

ASBTRACT

Enhancing the Reserved Graph Grammar (RGG) formalism, this paper introduces a size-increasing condition on the structure of graph grammars' productions to simplify the definition of graph grammars, and a general parsing algorithm to extend the power of the RGG parsing algorithm.

1. Introduction

Like textual programming languages that are usually equipped with proper formal syntax definitions and parsers, graphical or visual languages need the support of such mechanisms. This paper presents our recent work on an attempt to improve the RGG formalism [1] in two directions: simplify the design of an RGG, and enhance its expressive power. We will call this enhanced version RGG+.

The contribution of the RGG+ is twofold. One is to replace the multiple layers of labels with the usual two layers, i.e., terminal and non-terminal labels, and introduce a size-increasing condition to graph grammar productions to solve the membership problem. The size-increasing condition only imposes some weak restrictions on the structure of productions, and is more intuitive and easier to handle than the layer decomposition and the lexicographical order [2] used in the RGG. The other is to give a more general parsing algorithm that does not require its graph grammars to be confluent. This greatly enhances the parsing capability.

2. Notations and Concepts

Φ: An empty set.

Ω: A finite set of labels, consisting of two disjoint subsets, terminal label set Ω^T and non-terminal label set Ω^{NT}, i.e., $\Omega = \Omega^T \cup \Omega^{NT}$ and $\Omega^T \cap \Omega^{NT} = \Phi$.

$|\ |$: Cardinality of a set.

N: A node set consisting of a terminal node set N^T and a non-terminal node set N^{NT}, i.e., $N = N^T \cup N^{NT}$ with $N^T \cap N^{NT} = \Phi$.

f: A labeling function establishing a mapping $N \to \Omega$.

$p \coloneqq (L,R)$: A production consisting of a left L and a right R graphs over the same label set Ω.

$R(H,G)$: A set of redexes of graph G, which are sub-graphs of graph H.

$T(H,G',G,\tilde{H})$: Transforming graph H by replacing its sub-graph $\tilde{H} \in R(H,G)$ with G' to yield a new graph.

$H \mapsto^R H'$: R-application or reduction of a production $p \coloneqq (L,R)$ to graph H, namely $H' = T(H,L,R,\tilde{H})$.

$H \to^L H'$: L-application or derivation of a production $p \coloneqq (L,R)$ to graph H, namely $H' = T(H,R,L,\tilde{H})$.

$H \mapsto^* H_n$: A series of R-applications: $H \mapsto^{R1} H_1$, $H_1 \mapsto^{R2} H_2$, …, $H_{n-1} \mapsto^{Rn} H_n$ including the case $n = 0$ when $H = H_n$ and $H \mapsto H$.

$H \to^* H_n$: A series of L-applications: $H \to^{L1} H_1$, $H_1 \to^{L2} H_2$, …, $H_{n-1} \to^{Ln} H_n$ including the case $n = 0$ when $H = H_n$ and $H \to H$.

3. The Enhancements

This section inherits from the RGG the most basic concepts, such as graph element, graph, marking, isomorphism, redex, and graph transformations etc.

3.1 Definition of a RGG+ and its language

Definition 3.1 $gg \coloneqq (A,P,\Omega)$ is a graph grammar called *RGG+*, where A is an initial graph, P a set of graph grammar productions, Ω is a finite label set and can be further divided into two disjoint subsets, Ω^T and Ω^{NT}, for terminals and non-terminals respectively. For each production, $p = (L,R) \in P$, the following conditions must be satisfied:

- R is nonempty,
- L and R are over the same label set Ω, and

- the size of R must be no less than that of L, i.e., $|\, p.L.N \,| \leq |\, p.R.N \,|$; if they are equal, the number of non-terminal nodes in R must be more than that in L, i.e., $|\, p.L.N \,| = |\, p.R.N \,| \rightarrow |\, p.L.N^T \,| < |\, p.R.N^T \,|$.

Definition 3.2 Let $gg := (A, P, \Omega)$ be a RGG+, its *language* $\Gamma(gg)$ is a set of graphs that can be derived from the initial graph A and each graph node has one terminal label, i.e.,

$$\Gamma(gg) = \{G \mid A \mapsto^* G \wedge f(G.N) \subseteq \Omega^T\}.$$

3.2 Decidability

Theorem 3.1 For every RGG+ $gg := (A, P, \Omega)$ and for an arbitrary nonempty graph H ($H.N \neq \Phi$), it is decidable whether or not H is in $\Gamma(gg)$.

Proof: For a given graph H with a finite number of terminal nodes, namely $H.N = H.N^T$ being a finite set, the total number of graphs $\hat{H}$ with $f(\hat{H}.N) \subseteq \Omega$ and $|\, \hat{H}.N \,| \leq |\, H.N \,|$ must be finite under finite Ω. Considering a sequence of graphs

$$A = \hat{H}_0, \hat{H}_1, \hat{H}_2, ..., \hat{H}_{n-1}, \hat{H}_n = H$$

such that $\hat{H}_i.N^{NT} \neq \Phi$ and $|\, \hat{H}_i \,| \leq |\, \hat{H}_{i+1} \,|$ for $i = 0,1,..., n-1$ and $\hat{H}_i \neq \hat{H}_j$ if $i \neq j$, the number of such sequences without repetition is also finite. Thus we can enumerate all such sequences and check whether $\hat{H}_i \rightarrow^* \hat{H}_{i+1}$ ($i = 0,1,..., n-1$) holds for at least one of them. If so, then clearly $H \in \Gamma(gg)$, otherwise, $H \notin \Gamma(gg)$. □

4. Graph Parsing

The parsing algorithm of the RGG is very efficient in parsing any graph in polynomial time under the condition that the graph is derivable from the selection-free grammars [1]. In order to support the specification of those context-sensitive grammars that are not selection-free, we develop a more general parsing algorithm that attempts to search all possible parsing paths instead of just one as in the RGG.

4.1. A Parsing Algorithm

```
Parsing (Graph H , ProductionSet P )
{
        Initialization;
loop-1: while ( H ≠ A )
        {
           DELIMITER → redexStack;                    // push
Loop-2: for all p ∈ P
           {
              redexSet = FindRedexForRight( H , p.R );
loop-3:       for all redex ∈ redexSet;
              redex → redexStack;                     // push
           }
           redex ← redexStack;                        // pop
loop-4:    while (redex = DELIMITER)
           {
              H ← hostStack;                          // pop
              redex ← redexStack;                     // pop
              if (redex = NULL)
                 return("Invalid");
           }
           hostStack ← H ;                            // push
           H = RightApplication( H , p , redex);
        }
        return("valid");
}
```

In the above function, the R-application is the same as that in the RGG, while the search for redexes needs to be extended from searching for one redex to all redexes as follows.

```
FindRedexForRight(Graph H , Graph R );
{
   redexSet = Φ ;
   nodeSequence = orderNodeSequence( R );
   candidateSet = findNodeSequenceSet( H ,
                                       nodeSequence);
   for all candidate ∈ candidateSet
      redexSet = redexSet+match(candidate, H , R );
   return(redexSet);
}
```

The function orderNodeSequence(R) sequences the nodes in the right graph according to their labels' alphabetic order. The function findNodeSequenceSet(H , *nodeSequence*) finds all possible node sequences from the host graph, each of which is isomorphic to *nodeSequence*. Finally, the function match(*candidate*, H , R) checks whether a candidate in the host graph is a redex of the right graph, if so, the candidate is returned as a redex, otherwise, a null is returned.

4.2. Parsing Complexities

The parsing algorithm is no doubt more powerful but its performance may be seriously penalized because of the extremely large search space.

4.2.1. Time Complexity

Theorem 5.1 The time complexity of the parsing algorithm is $O((\frac{n}{r!})^h (h^h h!)^r)$, where h is the number of nodes in the host graph to be parsed, r is the maximal number of nodes in the right graphs of all productions, and n is the number of productions in the given RGG+.

Proof: According to the structure of the parsing algorithm, its maximal time complexity can be expressed as:

$$t = O(l_1(l_2(t_1 + l_3) + l_4 + t_2)),$$

where l_1 is the maximal number of iterations in the outmost loop-1, l_2 is the number of iterations in the first inner loop-2, l_3 is the number of iterations in the innermost loop-3, l_4 is the number of iterations in the second inner loop-4, and t_1 and t_2 are the time complexities of FindRedexForRight() and RightApplication() respectively.

We first consider l_2, which is in fact the number of productions, i.e., $l_2 = n = |P|$. Since $l_2 \cdot l_3$ is the total number of actions in pushing redexes into the redex stack and l_4 is the partial number of actions in popping redexes from the redex stack, l_4 should be no more than $l_2 \cdot l_3$, and thus can be ignored. Since l_3 is the number of redexes found in the host graph with respect to the right graph of a given production, the maximal number of redexes is the all possible node combinations C_h^r, thus $l_3 \le C_h^r = O(h^r)$. When considering l_1, the worst case is when the algorithm's answer is 'invalid', and all redexes found during parsing will enter the stack. Each of the redexes, when popped out of the stack, leads to an iteration of the outmost loop. Therefore, l_1 equals to the number of the redexes found.

An iteration of the outmost loop produces no more than nC_h^r redexes for n productions and performs one R-application. According to the size-increasing condition, each R-application would reduce the size of the derived host graph. Since there are at most h R-applications that may not reduce the host graph size and an R-application will reduce the host graph size by at least 1, the following derivations hold for l_1:

$$l_1 \le (nC_h^r)^h nC_{h-1}^r nC_{h-2}^r ....nC_{h-(h-r-1)}^r nC_{h-(h-r)}^r$$

$$= n^{2h-r}(\frac{h!}{(h-r)!r!})^h \prod_{u=1}^{h-r-1}\frac{(u+r)!}{r!u!}$$

$$= O((\frac{n}{r!})^{2h} h^{rh} \prod_{u=1}^{h-r-1} u^r)$$

$$= O((\frac{n}{r!})^h (h^h h!)^r)$$

As for t_1 and t_2, since the maximal possible number of selections of r nodes from h nodes is $A_h^r = h(h-1)...(h-r+1)$, the worst cases of searching for all redexes of a right graph in a given host graph must be $t_1 = O(h^r)$. Since t_2 is independent of h, it can be considered as $t_2 = O(1)$ that is bounded by a constant time.

Combining all the above discussions, we can finally obtain:

$$t = O((\frac{n}{r!})^h (h^h h!)^r). \qquad \square$$

4.2.2. Space Complexity

Theorem 4.2 The space complexity of the parsing algorithm is $O(h^{r+1})$, where h is the number of nodes in the host graph to be parsed, r is the maximal number of nodes in all the right graphs of productions.

Proof: Obviously the main space-consuming components are the redex stack and the host graph stack used in the parsing algorithm. We can therefore express the maximal space complexity as:

$$s = s_1 + s_2,$$

where s_1 is the space used by the redex stack and s_2 is the one by host graph stack. Without loss of generality, we can assume that the space taken by a redex is r and that by a host graph is h. Different from time complexity, the use of the stack space is not always increasing because pop operations would release space for reuse. Hence, the worst case is the maximal occupied space along the longest reduction path, and the following derivations hold for the redex stack and the host graph stack respectively.

$$s_1 \le r(hnC_h^r + nC_{h-1}^r + ... + nC_{h-(h-r)}^r)$$

$$= \frac{n}{(r-1)!}(\frac{hh!}{(h-r)!} + \sum_{u=0}^{h-r-1}\frac{(u+r)!}{u!})$$

$$= O(h^{r+1} + \sum_{u=0}^{h-r-1} u^r)$$

$$= O(h^{r+1});$$

$$s_2 = hh + (h-1) + ... + r = O(h^2).$$

Since $r \ge 1$, we can obtain:

$$s = O(h^{r+1}). \qquad \square$$

From the above analysis, we observe that the space complexity is bounded by a polynomial factor while the time complexity is extremely high. Hence, improving the time efficiency of parsing is our immediate future work.

References

[1] D. Q. Zhang, K. Zhang, and J. Cao, "A Context-Sensitive Graph Grammar Formalism for the Specification of Visual Languages", *The Computer Journal*, (44)3, pp.187-200, 2001.

[2] J. Rekers and A. Schürr, "Defining and Parsing Visual Languages with Layered Graph Grammars", *Journal of Visual Languages and Computing*, 8(1), pp.27-55, Feb. 1997.