Journal of Southeast University (English Edition) Vol. 24, No. 4, pp. 455 – 461 Dec. 2008 ISSN 1003—7985

Context-attributed graph grammar framework for specifying visual languages

Zou Yang[1] Zeng Xiaoqin[1] Han Xiuqing[1] Zhang Kang[2]

([1]College of Computer and Information Engineering, Hohai University, Nanjing 210098, China)

([2]Department of Computer Science, University of Texas at Dallas, Texas 75080-3021, USA)

Abstract: Since the specifications of most of the existing context-sensitive graph grammars tend to be either too intricate or not intuitive, a novel context-sensitive graph grammar formalism, called context-attributed graph grammar(CAGG), is proposed. In order to resolve the embedding problem, context information of a graph production in the CAGG is represented in the form of context attributes of the nodes involved. Moreover, several properties of a set of confluent CAGG productions are characterized, and then an algorithm based on them is developed to decide whether or not a set of productions is confluent, which provides the foundation for the design of efficient parsing algorithms. It can also be shown through the comparison of CAGG with several typical context-sensitive graph grammars that CAGG is more succinct and, at the same time, more intuitive than the others, making it more suitably and effortlessly applicable to the specification of visual languages.

Key words: visual language; graph grammar; context-attributed; parsing; confluence

Diagrammatic visual languages, such as the entity-relationship data model, control/data flow diagram model and CASE tools have been frequently used in database design and software engineering. Unlike a textual language whose syntax is always formally defined to facilitate efficient parsing algorithms, most of the diagrammatic ones are not equipped with formal syntactical definitions and parsers, even though a graph grammar can serve as a natural tool for this purpose. This is due to the fact that most of the existing graph grammars require expertise to design. To make a graph grammar framework practically useful, a simple and intuitive specification mechanism, strong expressive power and an efficient parsing algorithm are essential.

In the literature, several practically useful graph grammars and their parsing methods have been proposed, such as relational grammar[1], unification grammar[2], constraint multiset grammar[3], C-edNCE graph grammar[4], and adaptive star grammar[5], etc. However, these formalisms have difficulty in specifying some popular graphs, such as abstract syntax graphs for ER diagrams, and thus are limited in expressive power. In addition to the formalism of graph grammars, parsing efficiency is another major issue to which researchers have also devoted much effort[6-9].

Rekers and Schürr[6] developed the layered graph grammar(LGG) formalism in order to specify a wider range of visual languages. The LGG is context sensitive and thus highly expressive. By explicitly specifying all context elements on both sides of productions to address the embedding problem, the LGG provides an intuitive formalism without any embedding rule for users to easily define visual languages. However, its lexicographical order enforcement on each production for ensuring the decidability of the membership problem is intractable, and its parsing algorithm is intricate and has exponential time complexity.

Based on the LGG, Zhang et al.[10] proposed another context-sensitive formalism, reserved graph grammar(RGG), which is an improvement over the LGG in simplifying the specification and parsing. The RGG introduces a two-level node structure in the node-edge format to represent any given graph, and a marking mechanism to identify each context element with a unique label. It imposes a simple embedding rule to address the embedding problem, rather than explicitly specifying context elements in each production as in the LGG. Furthermore, the same authors developed a so-called selection-free parsing algorithm(SFPA) for the RGG with polynomial time complexity based on confluent graph grammars, and concluded that a graph grammar is confluent whenever its productions are confluent. The RGG, however, has the following weaknesses. The marking technique and the two-level node structure are not intuitive; i.e., it is not straightforward enough to create a production of node-edge format with its context elements properly marked. Moreover, even a comparatively simple set of productions may be non-confluent, and thus the SFPA cannot be applied to such a graph grammar.

The above observations motivate the work in this paper. The contribution of this paper includes two aspects. First, a new context-sensitive graph grammar formalism, called context-attributed graph grammar(CAGG), is proposed, which is more succinct and, at the same time, more intuitive than the existing ones. Second, several properties of the CAGG are characterized, and an algorithm that decides whether a set of productions is confluent is proposed, laying the foundation for employing efficient parsing algorithms.

1 Fundamental Concepts in the CAGG

A graph grammar generally consists of an initial graph and a set of graph productions, and thus defines a set of graphs which are generated by iteratively and arbitrarily applying the productions to the initial graph in different manners. To demonstrate how a CAGG works, a process flow diagram is introduced in Fig. 1 as an example and a CAGG specifying process flow diagram is given in Fig. 2.

Received 2008-01-21.

Biography: Zou Yang(1976—), male, lecturer, yzou@hhu.edu.cn.

Foundation items: The National Natural Science Foundation of China(No. 60571048, 60673186, 60736015), the National High Technology Research and Development Program of China(863 Program)(No. 2007AA01Z178).

Citation: Zou Yang, Zeng Xiaoqin, Han Xiuqing, et al. Context-attributed graph grammar framework for specifying visual languages[J]. Journal of Southeast University (English Edition), 2008, 24(4): 455 – 461.

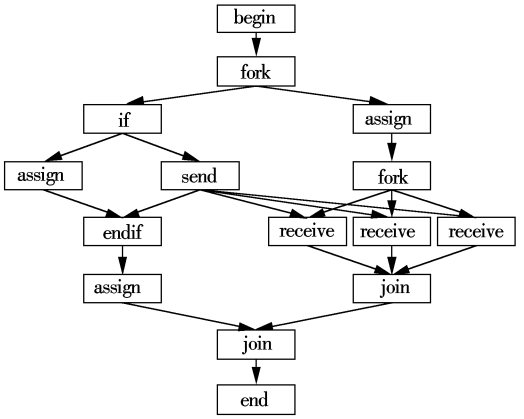

Fig. 1 A process flow diagram

A production, also called a rewriting rule, comprises two directed graphs each on one side of the definition symbol ":=", as shown in Fig. 2. The graph on the left (right) hand side is called the left (right) graph. For simplicity, we stipulate that each node in a production has two connecting points: one, on the top of it, and the other, at the bottom, are for edges directing to and from the node respectively. A production can be applied to given application graphs (called host graphs). A redex of a production is a subgraph in a host graph that is isomorphic to its left or right graph. When a redex of the left graph of a production occurs in a host graph, one can replace this redex by the right graph, which is called an L-application of this production. An R-application is the reverse replacement.

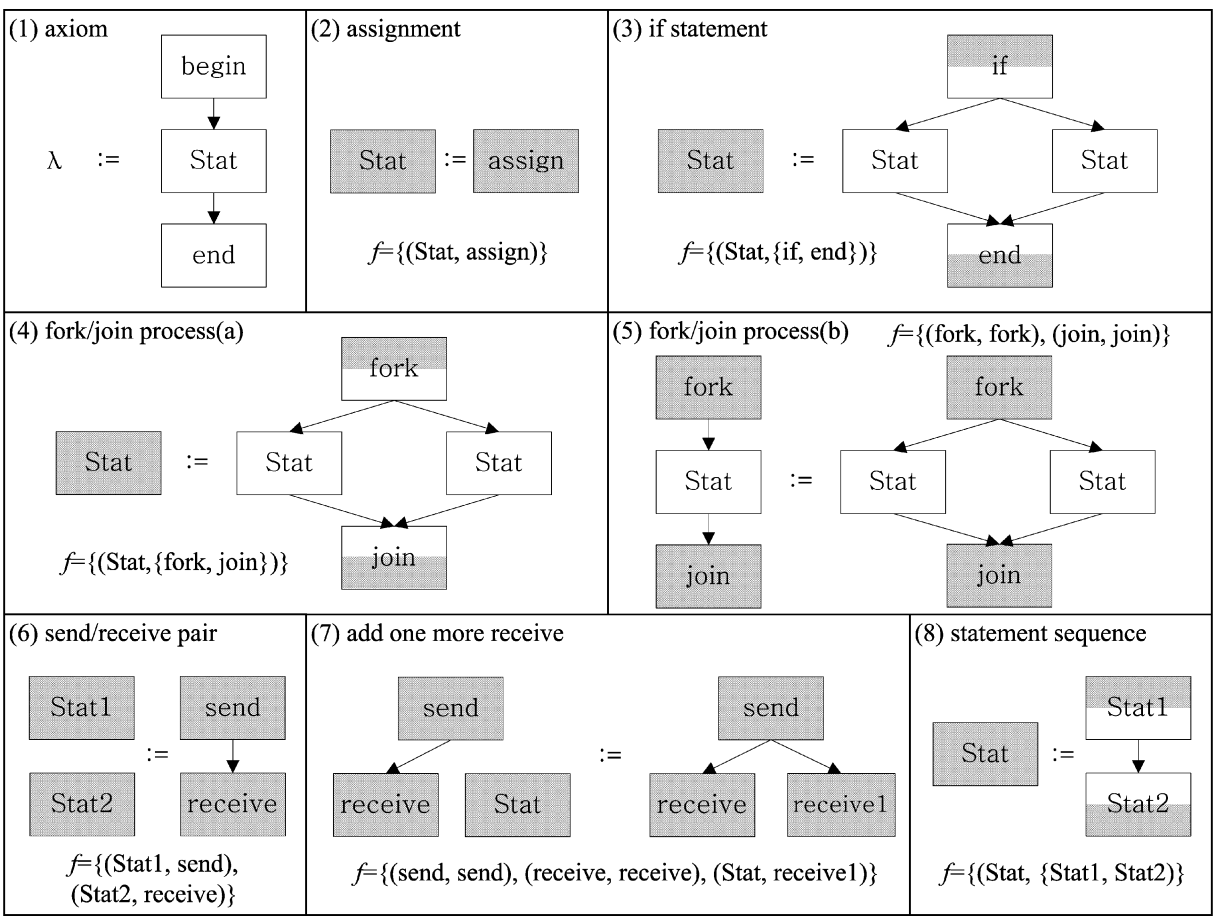

Fig. 2 A CAGG for process flow diagrams

The L-application defines the language of a graph grammar. The language comprises all possible graphs which have only terminal nodes and can be derived from the initial graph of the grammar. The R-application is used to parse a given graph in order to check whether it is a member of the language defined by the graph grammar. When conducting an L- or an R-application, we need a mechanism to establish relationships between the substitute of a redex and its surrounding elements, which is called the embedding problem[1,6].

During the process of embedding a graph into a host graph, it is necessary to avoid creating dangling edges. That is, the connecting positions on each node of a redex to or from which edges outside of it are directing should be preserved in the resulting host graph, since these edges are the bridges between the redex and its surrounding context elements. These connecting positions of interest can simply be divided into two categories: one, called the upper connecting point, is for edges arriving in the node; and the other, called the lower connecting point, is for edges leaving the node. Essentially, the connecting points on a redex, which encode the contextual information for embedding the redex into a host graph, can be collected from the left or right graph of the production to which they correspond. Using this fact, the CAGG introduces an attribute *ci* to each node in a production to specify the context information it carries (that is, the connecting points). The value of the *ci* can be *full*, *upper partial*, *lower partial* (hereafter abbreviated to *up* and *lp* respectively) and *null*. A node on a production with a *ci* evaluated to *up* (*lp*) preserves its upper (lower)

connecting point when it is applied, while a node with a *ci* evaluated to *full* keeps both points. A node with a *ci* evaluated to *null* does not contain any context information. As drawn in Fig. 2, the four types of nodes are partially shadowed in the upper half, partially shadowed in the lower half, completely shadowed and transparent. The nodes labeled "if" and "end" in production 3, the one labeled "Stat" in production 3 and that labeled "begin" in production 1 are corresponding examples. Production 3 also shows that the two connecting points in the node labeled "Stat" in the left graph are preserved, and correspond to the upper point in "if" and the lower point in "endif" in the right graph.

Then, an embedding rule, also called the dangling condition[6], is imposed on the CAGG requiring that if a node in the left or right graph of a production does not preserve a connecting point, and has an isomorphic node in a redex of the graph in a host graph, then all the edges connected to this point should be completely inside the redex.

This embedding rule guarantees that any application of a production will not create dangling edges.

In Fig. 3(a), the graph in the dashed rectangle is a redex of the right graph of production 5 but not of production 4 in that there is an edge outside of the dashed rectangle directed to the upper connecting point of "join", which contradicts production 4. The resulting graph after an R-application of production 5 is given in Fig. 3(b), which apparently is a redex of the right graph of both productions 4 and 5.

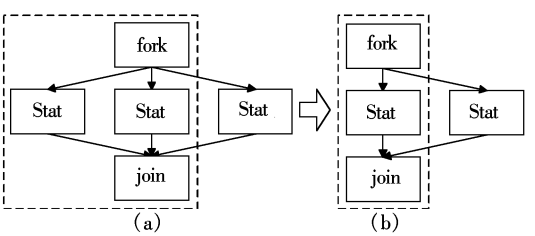

Fig. 3 An example of R-application

2 Formal Definition of the CAGG

This section first gives formal definitions of graphs, productions and redexes of productions in a host graph, and then defines the CAGG.

Definition 1 $G = (N, E, l_N, l_E, in, out)$ is a directed graph over two label sets Lb_N and Lb_E, where

- N and E are finite sets of nodes and edges, respectively;
- $l_N: N \to \mathrm{Lb}_N$ and $l_E: E \to \mathrm{Lb}_E$ are mappings from nodes and edges to their labels, respectively;
- $in(out): N \cup E \to E \cup N$ is a mapping that assigns each node the set of edges that are directed to(from) it or assigns each edge its target(source) node.

Definition 2 $G = (N, E, l_N, l_E, in, out, ci, nt)$ is an attributed directed graph over two label sets Lb_N and Lb_E, where N, E, l_N, l_E, in, out are defined as the above and

- $ci: N \to \mathrm{Ct} = \{full, up, lp, null\}$ assigns each node a piece of context information;
- $nt: N \to \{terminal, nonterminal\}$ assigns each node a node type.

Two operators $\vee$ (union) and $\wedge$ (intersection) are defined on Ct as: $up \vee lp = full$, $up \wedge lp = null$. The set of nodes in G containing context information (also called context nodes) is defined by $\mathrm{CI}(G) = \{n \in N \mid ci(n) \neq null\}$. The value *full* for context essentially indicates both *up* and *lp*, and thus a *full*-valued node n under some circumstances can also be viewed as two separate nodes which are evaluated to *up* and *lp*, respectively. For clarity, labels on nonterminal nodes are initially capitalized.

In the sequel, a directed (or an attributed directed) graph G will be compactly denoted as $G = (N, E)$, and N_G and E_G will be used to represent the sets of nodes and edges of G, respectively, whenever it is clear from the context.

Definition 3 Let G be an attributed directed graph and $\mathrm{CI}(G)$ the set of context nodes in G, a set $\{P_1, \ldots, P_n\}$ is an attribute partition on G, denoted by $\mathrm{Pr}(G)$, iff

- $P_i \subseteq \mathrm{CI}(G)$, $P_1 \cup \ldots \cup P_n = \mathrm{CI}(G)$, $P_j \cap P_k = \varnothing$, and $1 \leqslant |P_i| \leqslant 2$, where $1 \leqslant i, j, k \leqslant n; j \neq k$;
- If $P_i = \{n_1, n_2\}$ then $ci(n_1) = up$ and $ci(n_2) = lp$.

The principle underlying an attribute partition on a graph G is that the union of context information carried by every node in P_i can be assigned to a single node in another attributed directed graph; i. e., the context information collected from P_i must be *up*, *lp*, *full* or $\{up, lp\}$.

Definition 4 A production $p := (L, R, f)$ is a pair of attributed directed graphs over the same label sets Lb_N and Lb_E with a bijection $f: \mathrm{Pr}(L) \to \mathrm{Pr}(R)$ such that if $f(P_i) = Q_j (1 \leqslant i, j \leqslant n)$, then

- $|P_i| = 1 \Rightarrow |Q_j| = 1 \vee |Q_j| = 2$ and $|P_i| = 2 \Rightarrow |Q_j| = 1$;
- $ci(m_1) \vee \ldots \vee ci(m_k) = ci(n_1) \vee \ldots \vee ci(n_l)$, $1 \leqslant k, l \leqslant 2$;

where $\mathrm{Pr}(L) = \{P_1, \ldots, P_n\}$, $\mathrm{Pr}(R) = \{Q_1, \ldots, Q_n\}$, $P_i = \{m_1, \ldots, m_k\}$ and $Q_j = \{n_1, \ldots, n_l\}$.

The bijection f establishes a correspondence between the connecting points in the left graph L and the right graph R of a production, that is, R preserves the same context information as in L. For instance, $f(\{\mathrm{Stat}\}) = \{\mathrm{if}, \mathrm{and}\})$, written as $f = \{(\mathrm{Stat}, \{\mathrm{if}, \mathrm{and}\})\}$ in (3) of Fig. 2, means that the two connecting points on node "Stat" of L are preserved separately as the upper connecting point on "if" and the lower one on "end" of R. f is often omitted when the context information correspondence is clear from L and R. Hereafter, notations $p.L$ and $p.R$ are used to specify the left and right graphs of a production p.

Definition 5 A production $p := (L, R, f)$ is well-defined if for each connected component S of L or R, there is at least one node $n \in N_S$ such that $ci(n) \neq null$.

Definition 6 Two directed graphs $G_1 = (N_1, E_1, l_{N_1}, l_{E_1}, in_1, out_1)$ and $G_2 = (N_2, E_2, l_{N_2}, l_{E_2}, in_2, out_2)$ are isomorphic, denoted by $G_1 \cong G_2$, if there is a pair of bijections $f_1: N_1 \to N_2$ and $f_2: E_1 \to E_2$, satisfying

- $\forall n \in N_1 (l_{N_1}(n) = l_{N_2}(f_1(n)))$;
- $\forall e \in E_1 (l_{E_1}(e) = l_{E_2}(f_2(e)) \wedge f_1(in_1(e)) = in_2(f_2(e)) \wedge f_1(out_1(e)) = out_2(f_2(e)))$.

Definition 7 A subgraph S of a directed graph H is

a redex of an attributed directed graph G, denoted by $S \in \mathrm{Rd}(H, G)$, if there is a pair of bijections $f_1: N_G \rightarrow N_S$ and $f_2: E_G \rightarrow E_S$ such that

- $G \cong S$ under the two bijections f_1 and f_2;
- $\forall n \in N_G \ \forall e \in E_H((ci(n) = up \vee ci(n) = null) \wedge (out(e) = f_1(n)) \Rightarrow in(e) \in N_S)$;
- $\forall n \in N_G \ \forall e \in E_H((ci(n) = lp \vee ci(n) = null) \wedge (in(e) = f_1(n)) \Rightarrow out(e) \in N_S)$.

Definitions 4 and 7 guarantee that L- or R-application of a production to a host graph will never create dangling edges.

Definition 8 Let G_1 and G_2 be the left and right graphs in a production p with a bijection $g: \mathrm{Pr}(G_1) \rightarrow \mathrm{Pr}(G_2)$ where $\mathrm{Pr}(G_1) = \{P_1, \ldots, P_n\}$ and S is a redex of G_1 in a host graph H under two bijections f_1 and f_2 as defined above, the substitution of S in H by G_2 is the following process:

1) Delete S from H except for these nodes with $ci(n) \neq null$ (denoted by R_S);

2) Partition R_S into $\{V_1, \ldots, V_n\}$ such that $n \in V_i$ iff $n' \in P_i$ where $n = f_1(n')$ and $1 \leqslant i \leqslant n$, and then replace V_i in H with $g(P_i)$;

3) Glue G_2 onto H along $\bigcup_{i=1}^{n} g(P_i)$.

In the second step of the substitution, it is worth mentioning that when $|V_i| = 1$ and $|g(P_i)| = 2$, say, $V_i = \{n\}$ and $g(P_i) = \{n_1, n_2\}$ with $ci(n_1) = up$ and $ci(n_2) = lp$, those edges in H originally directed to and from n should be directed to n_1 and from n_2 respectively. This substitution relation between H and the resulting graph H' is written as $H' = \mathrm{St}(G_1, G_2, S, H)$. Since H' is completely dependent on S, this process can be succinctly described as $H \mapsto^S H'$.

Similar to a textual grammar, notations on derivation and reduction are necessary. By $H \mapsto^{S_1, \ldots, S_n} H'$ we denote a series of substitutions in the sequence of $S_1, \ldots, S_n$ from H to H', where $n \geqslant 0$. When the sequence of redexes is omitted, the process is compactly described as $H \mapsto^* H'$.

Definition 9 An L-application (or R-application) of a production $p: = (L, R, f)$ to a host graph H is a substitution $H' = \mathrm{St}(L, R, S, H)$ (or $\mathrm{St}(R, L, S, H)$), where $S \in \mathrm{Rd}(H, G)$.

Based on the above fundamental concepts, the definition of the CAGG is given as follows.

Definition 10 A context-attributed graph grammar (CAGG) is a tuple (A, P), where A is an initial graph (can also be viewed as a special case of a production with an empty left hand side λ, called an axiom) and P a set of well-defined productions. For each production $p := (L, R, f) \in P$, one of the following two conditions must be satisfied:

- The size of R is larger than that of L, i. e., $|N_L| < |N_R|$;
- If $|N_L| = |N_R|$, then N_R has more terminal nodes than N_L.

This definition imposes a syntactical constraint[11] on each production of a CAGG to ensure that its membership problem is decidable. In the sequel, we only focus on well-defined productions and simply refer to them as "productions".

Definition 11 The set of sentential forms of a CAGG $gg: = (A, P)$ is $\mathrm{Gr}(gg) = \{G \mid A \mapsto^* G\}$.

Definition 12 The language of a CAGG $gg: = (A, P)$, denoted as $L(gg)$, is a subset of $\mathrm{Gr}(gg)$ such that $L(gg) = \{G \in \mathrm{Gr}(gg) \mid \forall n \in N_G (nt(n) = terminal)\}$.

Similar to the conclusions drawn in Refs. [10 − 11], we have

Proposition 1 Given a CAGG $gg: = (A, P)$ and a nonempty graph H, whether $H \in L(gg)$ or not is decidable.

Proposition 2 Given a CAGG $gg: = (A, P)$ and a nonempty graph H, $H \in L(gg)$ iff there exists a sequence of redexes Rs such that $H \mapsto^{\mathrm{Rs}} A$.

Proposition 2 summarizes the relationship between membership decidability and R-applications by showing that a directed graph H being a member of a language defined by a CAGG can be determined by attempting a sequence of R-applications from H to the initial graph A. Conversely, L-applications can generate all the graph members of the language from A.

3 Graph Parsing

Graph parsing involves determining if a graph is syntactically well founded by reducing it according to a predefined graph grammar. The reduction is performed by conducting a sequence of R-applications. The process of parsing a graph H with respect to a graph grammar gg is as follows: Search H to find a redex of any production p in gg, and then perform an R-application of p to H; this process iterates until no redex can be found in the resulting graph, say H'. If H' is the initial graph of gg, the parsing process succeeds, indicating that H is a valid graph of gg. Otherwise, the above process repeatedly attempts other possible sequences of redexes. H is invalid only if it fails in all the possible attempts. The SFPA, the parsing algorithm for the RGG[10], can be efficiently performed in polynomial time under the assumption that the underlying grammar is confluent. A sufficient condition for deciding the confluence of graph grammars is also given in Ref. [10].

3.1 Properties

This subsection defines some general concepts, and concludes with several useful properties of the CAGG. Then, an algorithm is developed for deciding whether a set of productions is confluent.

Definition 13 A directed graph G is a merger of graphs A and B, if

- A and B are subgraphs of G;
- $\forall n \in N_G (n \in N_A \vee n \in N_B) \wedge \forall e \in E_G (e \in E_A \vee e \in E_B)$;
- $\mathrm{Cm}(A, B, G) = (N', E')$ is a non-empty graph;

where $N' = \{n \in N_G \mid n \in N_A \wedge n \in N_B\}$, $E' = \{e \in E_G \mid e \in E_A \wedge e \in E_B\}$.

The set of mergers of two graphs A and B is denoted by $\mathrm{Mrg}(A, B)$.

Definition 14 Let G be a merger of two directed graphs A and B, p_1 and p_2 two productions, and A and B isomorphic to $p_1.R$ and $p_2.R$, respectively. G is reducible with respect to p_1 and p_2 if $A \in \mathrm{Rd}(G, p_1.R) \wedge B \in \mathrm{Rd}(G, p_2.R) \Rightarrow \exists G_A, G_{AB}, G_B, G_{BA} (G \mapsto^A G_A \mapsto^B G_{AB} \wedge G \mapsto^B G_B \mapsto^A G_{BA} \wedge G_{AB} \cong G_{BA})$.

Noticeably, the common subgraph $\mathrm{Cm}(A, B, G)$ of A and B in G is isomorphic to a subgraph in $p_1.R$ and symmetrically in $p_2.R$, and we denote these two subgraphs as $\mathrm{Im}(\mathrm{Cm}(A, B, G), p_1.R)$ and $\mathrm{Im}(\mathrm{Cm}(A, B, G), p_2.R))$, respectively.

Theorem 1 Let G be a merger of two directed graph A and B, p_1 and p_2 two productions, and A and B isomorphic to $p_1.R$ and $p_2.R$, respectively. Then $A \in \mathrm{Rd}(G, p_1.R) \wedge B \in \mathrm{Rd}(G, p_2.R)$ iff

- $\forall n \in S(ci(n) = null) \Rightarrow in(n) \subseteq E_S \wedge out(n) \subseteq E_S$;
- $\forall n \in S(ci(n) = up) \Rightarrow out(n) \subseteq E_S$;
- $\forall n \in S(ci(n) = lp) \Rightarrow in(n) \subseteq E_S$

where $S \in \{\mathrm{Im}(\mathrm{Cm}(A, B, G), p_1.R), \mathrm{Im}(\mathrm{Cm}(A, B, G), p_2.R)\}$.

Proof Only if part. Consider the first case. Let $S = \mathrm{Im}(\mathrm{Cm}(A, B, G), p_1.R)$, $S' = \mathrm{Im}(\mathrm{Cm}(A, B, G), p_2.R)$ and $f_1: N_S \rightarrow N_{S'}$ and $f_2: E_S \rightarrow E_{S'}$ are a pair of bijections between them(please note $S \cong S'$). Assume $n \in N_S$ with $ci(n) = null$ and $e' \in E_{p_2.R}$ with $in(e') = f_1(n)$. Assume $e' \notin E_{S'}$, then $\neg \exists e \in E_S (f_2(e) = e')$, since S and S' are the common subgraphs of $p_1.R$ and $p_2.R$ while ignoring the attribute of context information on each node. So, $\neg \exists e \in E_{p_1.R} (f_2(e) = e')$. By definition 7, A is not a redex of $p_1.R$ because the edge in G which corresponds to e' is a dangling edge directing to A from $G \backslash A$. A contradiction occurs. Hence, $e' \in E_{S'}$, and $e \in E_S$ such that $f_2(e) = e'$. Likewise, one can conclude that $e \in E_S$ if $e' \in E_{p_2.R}$ and $\mathrm{out}(e') = f_1(n)$. The other two cases can be proved in a similar way.

If part. Suppose that A and B are isomorphic to $p_1.R$ and $p_2.R$ under two bijections $f_1: (N_A \cup N_B) \rightarrow (N_{R_1} \cup N_{R_2})$ and $f_2: (E_A \cup E_B) \rightarrow (E_{R_1} \cup E_{R_2})$. Assume $n \in N_A \wedge n' = f_1(n)$ such that $ci(n') = null \wedge \exists e \in E_{G \backslash A} (in(e) = n \overline{\vee} out(e) = n)$. Then $n \in \mathrm{Cm}(A, B, G)$, otherwise $e \in E_{G \backslash A}$ does not hold. Consequently, $n' \in Im(\mathrm{Cm}(A, B, G), p_1.R,)$. Let $e' = f_2(e)$. According to the premises, we have $in(n') \subseteq E_S \wedge out(n') \subseteq E_S$, and then $e' \in E_S$. Therefore, $e \in E_{\mathrm{Cm}(A, B, G)}$, which contradicts the assumption $e \in E_{G \backslash A}$. Hence, for any $n \in N_A$ such that $n' = f_1(n)$, $\neg (ci(n') = null \wedge \exists e \in E_{G \backslash A} (in(e) = n \overline{\vee} out(e) = n))$ holds. Similarly, $\neg (ci(n') = up \wedge \exists e \in E_{G \backslash A} (out(e) = n))$ and $\neg (ci(n') = lp \wedge \exists e \in E_{G \backslash A} (in(e) = n))$ can be concluded as well. By definition 7, $A \in \mathrm{Rd}(G, p_1.R)$. Symmetrically, we have $B \in \mathrm{Rd}(G, p_2.R)$.

Corollary 1 Let G be a merger of two directed graphs A and B, p_1 and p_2 two productions. If $A \in \mathrm{Rd}(G, p_1.R) \wedge B \in \mathrm{Rd}(G, p_2.R)$, then $\exists n \in N_S$ such that $ci(n) \neq null$, where S is defined as above.

Proof It is sufficient to consider only the case $S = \mathrm{Im}(\mathrm{Cm}(A, B, G), p_1.R)$. From definition 13, we obtain $|N_S| \geqslant 1$. Assume $n \in N_S$ with $ci(n) = null$, then $in(n) \subseteq E_S \wedge out(n) \subseteq E_S$ by theorem 1. Therefore, $V \subseteq N_S$, where $V = \{v \mid v = in(e_1) \vee v = out(e_2), e_1 \in out(n), e_2 \in in(n)\}$. Suppose $\neg \exists v \in V$ such that $ci(n) \neq null$(The opposite assumption directly completes the proof). Then, we iteratively conduct the above process until the sole case occurs: S is a connected component of $p_1.R$. According to definition 5, the conclusion holds.

Theorem 2 Let G be a merger of two directed graphs A and B. If G is reducible with respect to two productions p_1 and p_2, $A \in \mathrm{Rd}(G, p_1.R) \wedge B \in \mathrm{Rd}(G, p_2.R)$, and A' and B' are graphs isomorphic to $p_1.L$ and $p_2.L$ respectively, then $\exists G' (G'$ is a merger of A' and $B')$ such that $\mathrm{Cm}(A, B, G) = \mathrm{Cm}(A', B', G')$.

Proof It is clear from definition 13 that $\mathrm{Cm}(A, B, G) \subseteq A$ and $\mathrm{Cm}(A, B, G) \subseteq B$. Since G is reducible with respect to two productions p_1 and p_2, and $A \in \mathrm{Rd}(G, p_1.R) \wedge B \in \mathrm{Rd}(G, p_2.R)$, $\exists G_A, G_{AB}, G_B, G_{BA} (G \mapsto^A G_A \mapsto^B G_{AB} \wedge G \mapsto^B G_B \mapsto^A G_{BA} \wedge G_{AB} \cong G_{BA})$, by definition 14. So, $B \subseteq G_A$. Since $G_A = (G \backslash A) \cup A' = (B \backslash \mathrm{Cm}(A, B, G)) \cup A'$, $\mathrm{Cm}(A, B, G) \subseteq A'$. Therefore, $\mathrm{Cm}(A, B, G) \subseteq \mathrm{Cm}(A', B, G_A)$. Because $G \mapsto^A G_A$, the converse derivation $G_A \mapsto^{A'} G$ also holds. Then we obtain $\mathrm{Cm}(A', B, G_A) \subseteq \mathrm{Cm}(A, B, G)$ in the same way. Consequently, $\mathrm{Cm}(A', B, G_A) = \mathrm{Cm}(A, B, G)$. By the symmetric property, we have $\mathrm{Cm}(A', B, G_A) = \mathrm{Cm}(A', B', G_{AB})$. Hence, $\mathrm{Cm}(A, B, G) = \mathrm{Cm}(A', B', G_{AB})$.

This conclusion states that the prerequisite for the reducibility for any merger G of the right graphs of productions p_1 and p_2 is that, there is a merger G' of the left graphs of them such that $\mathrm{Cm}(p_1.R, p_2.R, G) = \mathrm{Cm}(p_1.L, p_2.L, G')$ when ignoring the context information on p_1 and p_2, which reveals the structural similarity between a pair of reducible productions.

Definition 15 Let P be a set of productions. P is confluent if for any $p_1, p_2 \in P$, A and B are directed graphs isomorphic to $p_1.R$ and $p_2.R$, respectively, and for each $G \in \mathrm{Mrg}(A, B)$, G is reducible with respect to p_1 and p_2.

Definition 16 Let $gg: = (A, P)$ be a CAGG, and $\mathrm{Gr}(gg)$ the set of sentential forms of gg. gg is confluent if $\forall G \in \mathrm{Gr}(gg) \wedge p \in P \wedge \forall S \in \mathrm{Rd}(G, p.R) \Rightarrow G \mapsto^S G' \mapsto^* A$.

The confluence property in Ref. [10] is also applicable in the CAGG, and is given as follows.

Theorem 3 Let $gg: = (A, P)$ be a CAGG. Then gg is confluent if P is confluent.

3.2 Algorithm IsConfluent

Based on these properties, an algorithm for deciding whether a set of productions is confluent is then developed. In the algorithm, the properties proved in theorem 1 and its corollary(denoted as C_1 and C_2 respectively),

and theorem 2 are exploited to filter out those mergers that do not simultaneously include redexes of productions p_i and p_j as early as possible, thus improving the efficiency. The algorithm is as follows:

Algorithm IsConfluent

Input: A set of productions $P = \{p_1, \ldots, p_n\}$;

Output: If P is confluent, then return "Yes", and "No" otherwise;

```
{
  for all p_i ∈ P, all p_j ∈ P, i ≠ j{//loop 1
  Cm = common subgraphs of p_i. R and p_j. R satisfying C_2;
  if( Cm ≠ null) {
    Transform each subgraph in Cm into a merger, and if it satisfies C_1, store it in Mrg;
    if( Mrg ≠ ∅) {
      Cp = (p_i. L ∩ p_j. L) ∩ (p_i. R ∩ p_j. R);
      if( Cp = null) return "No";
      else {
        for all S ∈ Mrg//loop2
          if¬ (S is reducible) return "No";
      }
    }
  }
  return "Yes";
}
```

The time complexity of algorithm IsConfluent is exponential in terms of the maximal number of nodes in the right graphs of the productions in P. In practice, however, the number of productions in a graph grammar and the number of nodes in a production's right graph are usually small. So, the exponential time complexity is actually not a crucial problem for the application of the algorithm.

4 Comparison with Related Graph Grammars

Similar to textual grammars, a context-sensitive graph grammar is more expressive than a context-free graph grammar. Here, comparisons are conducted between CAGG and other two graph grammars, LGG[6] and RGG[10], which are the most prominent context-sensitive graph grammars proposed so far.

As to the treatment of context information, the LGG views the common subgraph of the left graph and the right graph of a production as context elements, and introduces label wildcards to denote them. The explicitly specified context elements may increase the complexity of searching redexes in a host graph. Besides, it is difficult to exactly identify what each wildcard stands for in complex practical domains. However, the CAGG simply introduces an attribute to each node to implicitly identify its context information along with an embedding rule to resolve the embedding problem. We, therefore, believe that the CAGG is more succinct than the LGG.

The RGG introduces a two-level node structure where the first level is a super vertex and the second consists of several small vertices which are embedded in the first level. Both a vertex and a super vertex can be a connecting point of edges. With the node structure, an ordinary graph, when transformed into a node-edge format, looks complex. It is somewhat tricky to put suitable vertices into a node and appropriately arrange edges connecting them during the transformation. In addition, the RGG uses a marking mechanism to identify some vertices in a production with unique integers. These identified vertices along with the edges connected to them will be preserved when the production is applied. But which vertices should be marked and which should not are not so obvious to determine. CAGG introduces only an attribute for each node to specify its context information and can be directly applied to the specification of ordinary graphs. Therefore, compared with the RGG, the CAGG is more intuitive and succinct.

A graph production specified in the form of LGG, RGG, and CAGG, respectively, is shown in Fig. 4.

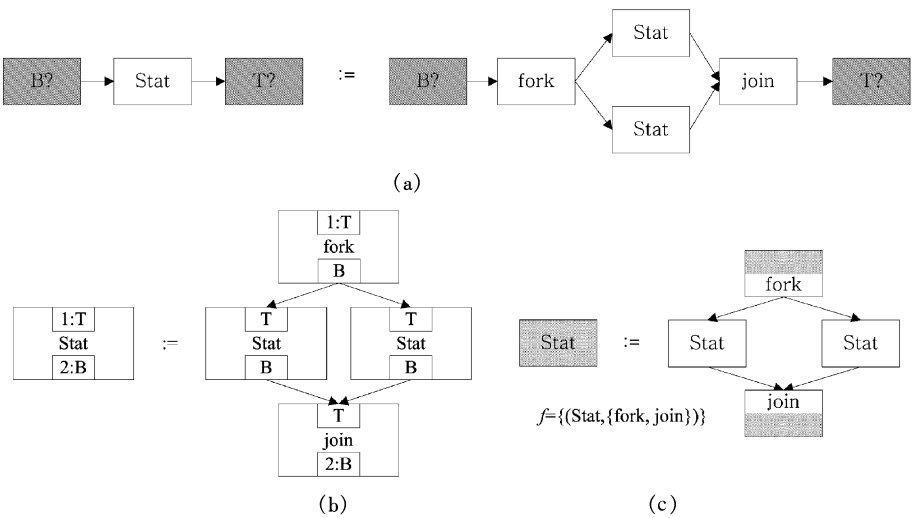

Fig. 4 A production comparison among LGG, RGG and CAGG. (a) An LGG production; (b) An RGG production; (c) A CAGG production

Strict comparison of expressive power among context-sensitive graph grammars is impractical, and no such conclusions have yet been drawn between LGG and RGG. An observation in this regard, however, has been made that CAGG can be applied to all the existing application cases specified by LGG and RGG[6, 8, 10-11].

5 Conclusion

This paper presents a novel context-sensitive graph grammar framework, called the CAGG. The CAGG possesses two qualities crucial to the application of graph grammar theories: succinctness and intuitiveness, making it effortless to specify context-sensitive graph grammars. Moreover, several properties of the CAGG are provided, and then an algorithm based on these properties is developed to decide whether a set of productions is confluent. These properties lay the foundation for further research on more efficient parsing algorithms especially for the CAGG.

References

[1] Ferrucci F, Tortora G, Tucci M, et al. A predictive parser for visual languages specified by relation grammars [C]//*Proc IEEE Symposium on Visual Languages*. St. Louis, Missouri, USA, 1994: 245 - 252.

[2] Wittenburg K, Weitzman L, Talley J. Unification-based grammars and tabular parsing for graphical languages [J]. *Journal of Visual Languages and Computing*, 1991, **2**(4): 347 - 370.

[3] Marriott K. Constraint multiset grammars [C]//*Proc IEEE Symposium on Visual Languages*. St. Louis, Missouri, USA, 1994: 118 - 125.

[4] Rozenberg G. *Handbook on graph grammars and computing by graph transformation: foundations*[M]. World Scientific, 1997: 1 - 94.

[5] Drewes F, Hoffmann B, Janssens D, et al. Adaptive star grammars [C]//*ICGT*2006, *LNCS* 4178. Berlin: Springer, 2006: 77 - 91.

[6] Rekers J, Schürr A. Defining and parsing visual languages with layered graph grammars [J]. *Journal of Visual Languages and Computing*, 1997, **8**(1): 27 - 55.

[7] Bottoni P, Taentzer G, Schürr A. Efficient parsing of visual languages based on critical pair analysis and contextual layered graph transformation [C]//*Proc IEEE Symposium on Visual Languages*. Seattle, Washington, USA, 2000: 59 - 60.

[8] Kong J, Zhang K. Parsing spatial graph grammars [C]//*Proc IEEE Symposium on Visual Languages and Human-Centric Computing*. Rome, Italy, 2004: 99 - 101.

[9] Minas M. Parsing of adaptive star grammars [C]//*Proc* 2*nd International Workshop on Graph and Model Transformation*. Brighton, UK, 2006, **4**: 1 - 15.

[10] Zhang D Q, Zhang K, Cao J. A context-sensitive graph grammar formalism for the specification of visual languages [J]. *The Computer Journal*, 2001, **44**(3): 187 - 200.

[11] Zeng X, Zhang K, Kong J, et al. RGG + : an enhancement to the reserved graph grammar formalism [C]//*Proc IEEE Symposium on Visual Languages and Human-Centric Computing*. Dallas, TX, USA, 2005: 272 - 274.

一个描述可视化语言上下文属性化的图文法框架

邹 阳[1] 曾晓勤[1] 韩秀清[1] 张 康[2]

([1] 河海大学计算机及信息工程学院,南京 210098)

([2] 德克萨斯大学达拉斯分校计算机科学系,美国德克萨斯 75080-3021)

摘要:针对目前已有的上下文相关图文法的描述规范过于复杂或不太直观,提出了一个新的上下文相关图文法的形式框架:上下文属性化的图文法 CAGG. 该文法将产生式的上下文信息刻画成相关结点的上下文属性来解决嵌入问题. 而且进一步分析了合流的 CAGG 产生式集合的基本特征,并基于此设计了合流产生式集合的判定算法,从而为构造高效的语法分析算法奠定了基础. 通过与已有上下文相关图文法的对比分析可知,CAGG 图文法的形式更为简洁和直观,因而更适于且更易于应用到可视化语言描述领域.

关键词:可视化语言;图文法;上下文属性化;语法分析;合流

中图分类号:TP301.2