Graph Grammars and Their Applications
By Girish Keshav Palshikar

Abstract

A striking contribution of computer science is a mathematical formulation of the concept of a language and its syntax. Now, a language of linear sentences is not the only way we communicate. Structured diagrams, which are pictures made up of discrete units connected together according to well-defined rules, are an integral part of human communication. This is primarily due to their intuitive appeal, richness and conciseness. Software developers use a host of diagrams (e.g., entity-relationship diagrams, data flow diagrams or state transition diagrams) to communicate with each other as well as with the users. Diagrams are ubiquitous in practically all human endeavours be they architectural plans, electrical circuits, CAD drawings, maps, geometrical patterns on clothes, structures of biological systems etc. A natural question is: can we formalise the notion and syntax of a pictorial language in which a symbol may have more than two neighbours (rather than just left and right neighbours as in a string)? Understanding such graphical, visual languages is the subject matter of graph grammars. Among the several ways invented to generalise the notions of string grammars, we look at two specific types, namely, NLC and NCE graph grammars. We demonstrate their use by specifying the syntax of some simple diagrams. We discuss issues in the theory of graph grammars. We outline our current work in graph grammars and conclude with other possible applications within TCS.
Graph Grammars and Their Applications
By Girish Keshav Palshikar

1 Introduction

A striking contribution of computer science is a mathematical formulation of the concept of a language and its syntax. An alphabet is a finite set of discrete symbols. A string over an alphabet is an ordered sequence (i.e., a linear chain) of the symbols from the alphabet. A formal language is a set of strings over a given alphabet. The mechanism of a formal grammar is used to specify the syntax of a language i.e., a grammar defines the characteristics of the strings in a language. A well-defined derivation procedure can show the syntactic correctness of a string in the language with respect to the grammar. A number of different grammar formalisms (like regular or context-free grammars) have been studied extensively.

Now, a language of linear sentences is important for communication among humans; but it is not the only way we communicate. Structured diagrams, which are pictures made up of discrete units connected together according to well-defined rules, are an integral part human communication. This is primarily due to their intuitive appeal, richness and conciseness. Architects, for example, use plans to communicate with civil engineers. Software developers use a host of diagrams (e.g., entity-relationship diagrams, data flow diagrams or state transition diagrams) to communicate among each other as well as with the users. Diagrams are ubiquitous in practically all human endeavours be they electrical circuits, CAD drawings, maps, geometrical patterns on clothes, structures of biological systems etc. A natural question is: can we formalise the notion and syntax of a pictorial language in which a symbol may have more than two neighbours (rather than just left and right neighbours as in a string)? Understanding such graphical, visual languages is the subject matter of graph grammars.

For simplicity, we ignore the geometrical information contained in a diagram (like co-ordinates, angles, curves etc.) and work with diagrams which can be represented as directed graphs having labels for vertices and edges. We also ignore visual information in a diagram; e.g., colours and placement. Among the several ways invented to generalise the notions of string grammars, we look at two specific types, namely, NLC and NCE graph grammars. We demonstrate their use by specifying the syntax of some simple diagrams. We discuss issues in the theory of graph grammars which unfortunately is not as well-developed nor as well-organised as that of string grammars. We outline our current work and conclude with other possible applications within TCS.
In this section, we consider undirected, finite, vertex-labelled graphs; let $H(\Sigma)$ denote the set of all undirected, finite graphs each of whose vertex is labelled with some symbol from the alphabet $\Sigma$. An node-label controlled (NLC) graph grammar is a tuple $G = (T, N, P, C, Z)$ where $T$ and $N$ are terminal and non-terminal alphabets resp., $P$ is a set of graph productions, $C \subseteq \Sigma \times \Sigma$ is the connection relation where $\Sigma = T \cup N$ and $Z \in H(\Sigma)$ is the axiom or start graph. Each production in $P$ is of the form $S \rightarrow G$ where $S$ is a non-terminal from $N$ and $G \in H(\Sigma)$.

As an example, we give an NLC grammar for binary trees in which all internal vertices are labelled by $b$ and all leaves are labelled by $a$. Let $G1 = (\{a, b\}, \{A\}, P, \{(a,b), (b,b)\}, A)$ where the productions are as follows:

- $A \rightarrow a$
- $A \rightarrow A \bullet b \bullet A$

Each derivation step which creates a derived graph from a source graph proceeds as follows:

- Select a mother node (i.e., a vertex having a non-terminal label $X$) in the source graph.
- Mark all vertices adjacent to the mother node as the embedding area in the source graph.
- Obtain the rest graph by deleting the mother node and all edges incident on it.
- Select a production where the LHS is $X$ and $g$ is the graph on the RHS.
- Add a copy of $g$ (called the daughter graph) to the rest graph.
- For each connection $(x,y)$ in $C$, connect each vertex labelled $x$ in the daughter graph to each vertex in the embedding area which is labelled $y$.

For graphs $\alpha, \beta \in H(\Sigma)$, we say that $\alpha$ directly derives $\beta$, written $\alpha \Rightarrow \beta$, if $\beta$ is created from $\alpha$ using the above procedure.

The figure above shows the details in a derivation step for the given NLC grammar.

The language $L(G)$ generated by an NLC grammar $G$ is the set of all terminal-labelled graphs $g$ in $H(T)$ such that $g$ is derivable from the axiom $Z$. The problem of determining whether an arbitrary graph $g$ in $H(T)$ belongs to $L(G)$ is in NP. Also, no automata-like device is known to accept $L(G)$. 

The language $L(G)$ generated by an NLC grammar $G$ is the set of all terminal-labelled graphs $g$ in $H(T)$ such that $g$ is derivable from the axiom $Z$. The problem of determining whether an arbitrary graph $g$ in $H(T)$ belongs to $L(G)$ is in NP. Also, no automata-like device is known to accept $L(G)$.
It is easy to see that the NLC formalism can be restricted in several ways. For example, the 
regular NLC (RNLC) graph grammar allows only productions of the form

\[ X \rightarrow a \quad \text{or} \quad X \rightarrow aY \]

where \( a \) is a terminal and \( X, Y \) are non-terminals. NLC grammars can also be extended; e.g., by 
allowing directed edges. However, a major limitation of the NLC grammar is that the connection 
relation is global and so ignores local situations for embedding the daughter graph. We now 
consider a graph grammar which allows a separate local connection relation with each production.

3  

\textbf{NCE Graph Grammar}

A \textit{neighbourhood controlled embedding (NCE) graph grammar} is a tuple \( G = (T, N, P, Z) \) where \( T, \) 
\( N \) and \( Z \) are as for NLC. Each \textit{graph production} in \( P \) is of the form \( S \rightarrow G, C \) where \( S \) is a 
non-terminal from \( N \), \( G \in \mathcal{H}(\Sigma) \) and \( C \subseteq N_G \times \Sigma \) is the \textit{local connection relation} where \( N_G \) is the set of 
vertices in \( G \). For example, the following graph grammar generates polygonal (or circuit) graphs. 
Here, \( T = \{a, b\} \) and \( N = \{ X \} \). The connection relation for each production is shown in braces. The 
numbers refer to the ID of a vertex in the graph on the RHS of the production. \( Z \) is the axiom.

\[
\begin{align*}
X &\rightarrow b \quad \{ (1,a), (1,b) \} \\
X &\rightarrow bX \quad \{ (1,b), (2,a) \} \\
Z: &aX
\end{align*}
\]

A derivation of a pentagon is shown below. The derivation step in NCE grammar is similar to that in 
NLC grammar except that at each step the rest graph is connected to the vertices in the 
embedding area as per the local connection relation in the production. If the connection relation 
contains a tuple of the form \((i,x)\) then the vertex \( i \) in the daughter graph is connected to each 
vertex in the embedding area which has the label \( x \).

\[
\begin{align*}
\begin{array}{c}
\text{a} \\
\text{X}
\end{array} \quad \Rightarrow \quad \begin{array}{c}
\text{b} \\
\text{a} \quad \text{X}
\end{array} \\
\quad \Rightarrow \\
\begin{array}{c}
\text{b} \\
\text{a} \quad \text{X}
\end{array} \quad \Rightarrow \\
\begin{array}{c}
\text{b} \quad \text{b} \\
\text{a} \quad \text{a} \quad \text{b}
\end{array}
\end{align*}
\]

4  

\textbf{Applications, Tools and Current Work}

Essentially, NLC and NCE graph grammars use node rewriting. Graph grammars based on 
techniques other than node rewriting have also been suggested. For example, edge labelled 
controlled graph grammars associate a terminal or a non-terminal label with an edge; then the
edge is removed and rewritten according to a production rule. Other approaches to graph grammars are based on hyperedge replacement, algebraic graph rewriting etc. Some graph grammars (e.g., map grammars) allow explicit representation of geometric information.

However, a number of theoretical issues remain undiscovered in the theory of graph grammars. For example, little work is done on automata-like devices which accept graph languages. There is no clear characterisation of a Chomsky-like hierarchy of graph grammars in their increasing power. There are few results about the comparative expressive powers of various approaches to graph grammars. Closure and decision properties of graph grammars are not well understood.

Yet, graph grammars offer a rich modelling formalism. They have been used to model sequences of cell divisions, structure of biological organisms in their life cycles, understanding of electrical circuits etc. In software, they have been used in CAD object models, specification of the structure of massively parallel programs, diagram editors, visual languages and in pattern recognition.

A number of tools are available for graph grammars. GraphEd is a graphical tool which allows creation of graph grammars (in NLC and NCE formalisms) and manipulation of graphs according to the syntax specified by a graph grammar. Plexus is a tool for analysis of graph grammars; e.g. it can be used to check if a given graph grammar contains a graph satisfying some property $\Pi$. Proceedings of a series of International Workshops on Graph Grammars and Their Applications contain most of the work about graph grammars; see [Ehrig 1987], [Ehrig 1991], [Cury 1995].

Availability of a reasonably expressive graph grammar along with an efficient parsing algorithm is likely to boost the use of graph grammars. My current research deals with the possibility of a DCG-style Prolog parser for ed-NCE graph grammars. I hope to use this tool as a specification mechanism for diagram syntax, process programming and representation and manipulation of knowledge in the business applications. If you are interested in applying graph grammars in your projects, please feel free to contact me.
Selected Bibliography

