Computer Science Review 28 (2018) 178–203

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Survey

Graph grammars according to the type of input and manipulated data: A survey

Saadia Albane [a], Hachem Slimani [a,*], Hamamache Kheddouci [b]

[a] LIMED Laboratory, Computer Science Department, University of Bejaia, 06000 Bejaia, Algeria
[b] Université de Lyon, CNRS, Université Lyon 1, LIRIS, UMR5205 F-69622, France

ARTICLE INFO

Article history:
Received 20 June 2016
Received in revised form 21 March 2018
Accepted 3 April 2018

Keywords:
Graph grammar
Type of input and manipulated data
Type of generated graph
Big Data
Cloud computing
Application

ABSTRACT

Graph grammars which generate graphs are a generalization of Chomsky grammars that generate strings. During the last decades there has been a remarkable development of graph grammars. Due to their wide diversity of applications, graph grammars have received a particular attention from many scientists and researchers. There has been applications of graph grammars in several areas such as pattern recognition, data base systems, biological developments in organisms, semantics of programming languages, compiler construction, software development environments, etc. In the literature, in some surveys, graph grammars have been studied and classified according to some criteria such as: parallel or sequential applicability of rules, embedding mechanism, type of generated graphs, etc. In addition to this, as data play an important role more and more in different domains, we survey in this paper the vast field of graph grammars by classifying them according to three criteria: the number of manipulated data (single or multiple types), the nature of data (structured or unstructured), and finally the kind of data (images, graphs, patterns, etc.). In particular, we consider that a graph grammar is well defined by five components instead of four, namely: type of generated graphs (T_G), a start graph (Z), a set of production rules (P), a set of additional specifications of the rules (A), and the criterion that we additionally consider which is the type of input and manipulated data (T_D). This proposed formalism, especially with the added fifth component, may serve to overcome some issues related to Big Data and Cloud Computing domains.

Contents

* Corresponding author.
E-mail addresses: saadialbane@gmail.com (S. Albane), haslimani@gmail.com (H. Slimani), hamamache.kheddouci@univ-lyon1.fr (H. Kheddouci).

https://doi.org/10.1016/j.cosrev.2018.04.001
1574-0137/

Table of symbols

Symbol	Meaning
GG	Graph Grammar.
V	Set of vertices.
E	Set of edges.
s_V	Assigns an edges source vertex,
t_V	Assigns an edges target vertex,
l_V	Assigns vertex labels.
l_E	Assigns edge labels.
E_D	Set of directed edges.
E_{ND}	Set of undirected edges.
T_D	Type of input and manipulated data.
T_G	Type of generated graph.
A	Set of additional specifications of the productions.
Σ	Set of vertex labels.
Σ_N	Set of nonterminal vertex labels.
Σ_T	Set of terminal vertex labels.
Γ	Set of edge labels.
Γ_N	Set of nonterminal edge labels.
Γ_T	Set of terminal edge labels.
P	Set of production rules.
Z	Initial graph or axiom.
I	Input vertices.
O	Output vertices.
$\mathcal{E}$	Embedding of the production rule.
WG	Web Grammar.
$APGG$	Attributed Programmed Graph Grammar.
GGA	Graph Grammar with Application conditions.
$nPCEGG$	node-replacement Graph Grammar with Path-Controlled Embedding.
$CFGG$	Context Free Graph Grammar.
$DNLCGG$	Directed Node Label Controlled Graph Grammar.
$NLCGG$	Node Label Controlled Graph Grammar.
$1DNCEGG$	Directed Graph Grammar with Neighborhood-Controlled Embedding and singleton left-hand sides.
$NCEGG$	Neighborhood Controlled Embedding Graph Grammar.
$eNCEGG$	Graph Grammar with Neighborhood Controlled Embedding and dynamic edge relabeling.
$neNCEGG$	Node replacement Graph Grammars with dynamic node relabeling.
$NRGG$ or $C\text{-}edNCGG$	Node Replacement Graph Grammar.
$R(A\text{-}DNLCGG)$	Relabeling of an Apex DNLC Graph Grammar.
$ADGG$	Attribute Dependency Graph Grammar.
$TAGG$	Tree Adjoining Graph Grammar.
$ASGG$	Adaptive Star Graph Grammar.
$edNCEGG$	Neighborhood Controlled Embedding with edge-labeled and directed Graph Grammar.
$HRGG$	Hyperedge Replacement Graph Grammar.
$ERGG$	Edge Replacement Graph Grammar.
$GGPD$	Graph Grammar for Package Diagrams.
$ACSGG$	Attributed Context-Sensitive Graph Grammar.
TGG	Triple Graph Grammar.
$PrGG$	Pair Graph Grammar.
PGG	Parsing Graph Grammar.
$cNRGG$	conditional Node Replacement Graph Grammar.
$POcNRGG$	Partially Ordered conditional Graph Grammar.

1. Introduction and motivation

As a generalization of Chomsky grammars which generate strings, graph grammars that generate graphs were first introduced, under the name of *Web Grammars* [1], to deal with picture processing problems. During the last decades, graph grammars, that are graph rewriting systems, have received a particular attention from many scientists and researchers because of their wide diversity of applications in several areas such as pattern recognition, data base systems, modeling objects and structure dynamics, biological developments in organisms, extraction of graph queries, semantics of programming languages, compiler construction, software development environments, bidirectional transformations, etc. Some of such applications can be found in [2–14]. In contrast to string grammars, and due to the properties of graphs, graph grammars represent a suitable formalism for describing structural manipulations of multidimensional data. The main principle of derivation in a graph grammar consists of recursively replacing from a given data graph a subgraph by another one, to generate terminal graphs representing the corresponding language.

The development process of the components that constitute basic elements of graph grammars has evolved over time. Initially, Pfaltz and Rosenfeld [1] used three components to represent web grammars, that are "phrase-structure" grammars which define languages whose sentences are directed graphs with symbols at their vertices, namely $WG = (\Sigma, Z, P)$ such that: Σ is a set of terminal and nonterminal symbols corresponding to the vertex labels; Z is a set of initial subgraphs; and P is a set of rewriting rules where each rule is mainly composed of a left hand side representing the subgraph to be substituted, and a right hand side corresponding to the subgraph to be inserted. Since then, for considerations of graph grammar applications in different fields, other components are introduced and involved in the definition of graph grammars, such as: Γ which is a set of edge labels; A', B' which are finite sets of vertex and edge attributes; C which is a control diagram over a set of productions P; $A(p) = (AP(p), AN(p))$ which is an application condition constituted of a positive (resp. negative) one $AP(p)$ (resp. $AN(p)$), etc. In this setting, for instance, by involving Γ, A', B' and C, Bunke [15] introduced *Attributed Programmed Graph Grammar* which generates undirected graphs with labeled vertices and edges over (Σ, Γ) for the interpretation of schematic diagrams. Furthermore, Fotso et al. [4] (resp. Baumann [16]) used this graph grammar for generating the diversity in a products family (for representing a-priori knowledge about common music notation). On the other hand, by involving $AN(p)$, Habel et al. [7] defined *Graph Grammar with Application Conditions* for the specification of a lift control system. Moreover, Machado et al. [9] investigated the modeling of dynamic systems by means of second-order graph grammars. In particular, they proposed a notion of rule-based modification of graph rules based on two concepts that are double-pushout rewriting and negative application conditions. Recently, De la para and Dean [17] proposed a formalism of four main elements which gathers all the above components to represent graph grammars, that is $GG = (T_G, Z, P, A)$ such that: T_G is a type of generated/recognized graphs over Σ and/or Γ, which can be directed or undirected, with labeled vertices and/or edges; Z is a start graph; P is a set of production rules; and A is a set of additional specifications of the rules which extends the manner of applying the production rules, such as: the applied embedding mechanism $\mathcal{E}$, the associated application conditions $A(p)$, the corresponding vertex and edge attributes A', B', the priority of application rules if any where the order can either be specified explicitly by using a control flow mechanism, or it can be given implicitly by causal dependencies of rule applications [18], etc.

As data volumes grow exponentially, one of the questions that might arise is: "Is the formalism of graph grammars proposed by De la para and Dean [17] sufficiently adequate to take into consideration of Big Data characteristics and requirements?". Big Data represent a huge volume of both structured and unstructured data which is so vast that it is hard to process them using the conventional methods. Specifically, these large amounts