Theoretical Computer Science 583 (2015) 40–50

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Node replacement graph grammars with dynamic node relabeling

Changwook Kim *, Mahiro Ando

School of Computer Science, University of Oklahoma, Norman, OK 73019, USA

ARTICLE INFO

Article history:
Received 27 September 2013
Received in revised form 23 January 2015
Accepted 21 March 2015
Available online 1 April 2015
Communicated by Z. Esik

Keywords:
Formal languages
Graph grammars
Node replacement system
Dynamic node relabeling

ABSTRACT

The paper introduces node replacement graph grammars with dynamic node relabeling as an extension of the eNCE families of graph grammars. Initiated by NLC grammars in which graph embedding is based on node labels only, node distinction of the embedded graph extended NLC grammars to NCE grammars and dynamic edge relabeling further extended NCE grammars to eNCE grammars. Now, dynamic node relabeling, the node counterpart of the dynamic edge relabeling, is introduced as the next extension, which we call neNCE. The paper analyzes basic language-theoretical properties of the neNCE system.

Published by Elsevier B.V.

1. Introduction

Graph grammars are well-investigated systems that describe languages of node- and/or edge-labeled graphs using graph rewriting. The node replacement system is one of the most successful models of graph grammars. The idea is simple: replace a single node with a graph in one step and repeat it until a graph of interest is generated. The rules of replacement are specified by a grammar's productions.

Along this line of study, Janssens and Rozenberg [8] introduced the node-label-controlled (NLC) grammars, and various extensions of this system have appeared in the literature. In the NLC grammars, the replacement of a graph is based only on the label of the replaced node and new connections are established based on the node labels of the embedded graph and the neighborhood of the replaced node. Janssens and Rozenberg [10] extended the NLC grammars to neighborhood-controlled embedding (NCE) grammars, which allow to distinguish nodes in the embedded graph, regardless of the labels, in order to establish connections. The eNLC and eNCE grammars ("e" for edge relabeling) [5,11], which involve dynamic edge relabeling, are additional extensions of the NLC and NCE grammars, respectively. In the eNLC and eNCE grammars, edges are also labeled and modified dynamically in each derivation step.

These graph grammars are powerful enough to describe PSPACE-complete languages, but do not describe all recursively enumerable string languages (because their languages are in PSPACE) [5]. Our motivation of the present paper is to introduce a new extension, called dynamic node relabeling, to give the Turing-complete power to the node replacement system. We shall call this system neNCE, "n" for node relabeling. Known systems that can simulate Turing machines include the handle NLC grammars of Main and Rozenberg [16], the edge-label controlled grammars of Main and Rozenberg [17], the handle-rewriting hypergraph grammars of Courcelle et al. [2] and the HRNCE hypergraph grammars of Kim and Jeong [15]. In particular, the HRNCE system was extended from eNCE with the goal of generating hypergraphs using a simple rewriting

* Corresponding author.
E-mail address: dalmaji@gmail.com (C. Kim).

http://dx.doi.org/10.1016/j.tcs.2015.03.040
0304-3975/Published by Elsevier B.V.

mechanism and of achieving the Turing-complete power. We believe that the neNCE system introduced in the present paper is the first node replacement system in the literature that possesses the Turing-complete description power.

Not only the extensions but also reasonable restrictions of the NLC grammars have been studied in the literature. For instance, in boundary NLC (B-NLC) grammars of Rozenberg and Welzl [18], any two nodes with nonterminal labels are not allowed to be adjacent. Linear NLC (Lin-NLC) grammars of Engelfriet and Leih [3] are even more restricted systems, in which any sentential form may contain at most one nonterminal node. These restrictions can be applied to the NCE, eNLC and eNCE grammars as well and a complete hierarchical relation among these systems was given in [12]. The main feature of such restrictions is that they are *confluent* (or context-free) [1], i.e., the order of applying production rules in a derivation has no influence on the resulting graph, and this implies easier handling of such systems. For example, the class B-eNCE is in NP [5]. We shall define similar confluent subsystems of the neNCE system, such as 3-separated and linear grammars, and analyze their basic properties. For example, it will be shown that the 3-separated neNCE grammars have the same power as the 3-separated eNCE grammars (thus, the power of dynamic node relabeling is subsumed by the power of dynamic edge relabeling in this case) and they have useful normal forms such as Chomsky and Greibach normal forms. Hierarchical relations among the neNCE families of graph languages will also be presented.

The paper is organized as follows. Section 2 contains basic definitions and notations. Section 3 defines the neNCE system and its subsystem nNCE (that utilizes dynamic node relabeling but not dynamic edge relabeling) and proves that nNCE grammars can describe all recursively enumerable string languages. Section 4 defines a few basic confluent neNCE subsystems and proves hierarchical relations (equivalence, proper inclusion and incompatibility) among the neNCE families of graph languages. Section 5 proves that confluent subsystems of the neNCE system defined in the present paper have Chomsky and Greibach normal forms, similar to the B-eNCE and Lin-eNCE cases.

2. Preliminaries

This section contains basic notations and definitions which will be needed in this paper. The mathematical notations discussed in this section follow general usage found in the context of formal language theory and graph theory.

For a set $\mathcal{A}$, its *cardinality* is denoted by $|\mathcal{A}|$, and its *power set*, or the set of all its subsets, is denoted by $2^{\mathcal{A}}$. The *empty set* is denoted by $\emptyset$. An *alphabet* is a finite set of symbols.

Let Σ and Γ be alphabets (of node and edge labels). A *graph over* Σ *and* Γ is a system $H = (V, E, \phi)$, where V is a finite set of *nodes*, E is a finite set of *edges* of the form $(\{u, v\}, \lambda)$, where $u, v \in V$, $u \neq v$ and $\lambda \in \Gamma$, and $\phi : V \to \Sigma$ is a *node labeling function*. An edge $(\{u, v\}, \lambda)$ will be simply denoted by (u, λ, v) or (v, λ, u) and the set $\{\lambda \mid (u, \lambda, v)$ is an edge in $H\}$ is denoted by $\psi_H(u, v)$. The three components of H are written as V_H, E_H and ϕ_H, respectively. As a special case, if $|\Gamma| = 1$ then H is called a *graph over* Σ. (Intuitively, a graph over Σ does not have edge labels.) The *empty graph*, which has no node and no edge, is denoted by Λ.

Let H be a graph over Σ and Γ. Then two nodes $u, v \in V_H$ are *adjacent* if (u, λ, v) is an edge for some λ; u is called *incident* to this edge, and vice versa. The set of all nodes adjacent to the node u in H is denoted by $\mathrm{adj}_H(u)$. The *context set* of a node u in H, denoted by $\mathrm{context}_H(u)$, is the set $\{(\psi_H(u, v), \phi_H(v)) \mid v \in \mathrm{adj}_H(u)\}$.

A sequence $p = (v_1, v_2, \ldots, v_r)$, $r \geq 2$, of distinct nodes in V_H is a *path* between v_1 and v_r if v_i and v_{i+1} are adjacent for all $i \in \{1, 2, \ldots, r-1\}$. If p is a path, $r \geq 3$, and v_1 and v_r are adjacent, then $(v_1, v_2, \ldots, v_r, v_1)$ is a *cycle*. H is *connected* if there is a path between each pair of its nodes. H is a *tree* if it is a connected graph without any cycle. H is a *chain* if it is a tree which is in fact a path.

Let H_1 and H_2 be graphs over Σ and Γ. Then H_1 and H_2 are *isomorphic* if there is a bijection $h : V_{H_1} \to V_{H_2}$ such that for all $u \in V_{H_1}$, $\phi_{H_1}(u) = \phi_{H_2}(h(u))$ and $(u, \lambda, v) \in E_{H_1}$ for any $v \in V_{H_1}$ and any $\lambda \in \Gamma$ if and only if $(h(u), \lambda, h(v)) \in E_{H_2}$. H_2 is an *isomorphic copy* (or simply *copy*) of H_1 if H_1 and H_2 are isomorphic.

The set of all graphs over Σ and Γ is denoted by $GR_{\Sigma,\Gamma}$ and that over Σ is denoted by GR_{Σ}. Any subset of $GR_{\Sigma,\Gamma}$ or GR_{Σ} is called a *graph language*.

For a string $w = a_1 a_2 \cdots a_n$ of length n, the *oriented chain of* w, denoted by o-chain(w), is an edge-unlabeled chain with $n+1$ nodes labeled sequentially by $a_1, a_2, \ldots, a_n, \$$, where $\$$ is a special symbol to identify the orientation. For a string language L, its equivalent graph language is o-chain$(L) = \{\text{o-chain}(w) \mid w \in L\}$.

3. Dynamic node relabeling

This section introduces our new graph grammar system neNCE, that extends eNCE, and its subclass nNCE corresponding to NCE. We shall then prove that these grammars can generate all recursively enumerable string languages, by simulating an arbitrary Turing machine on chains.

Definition 3.1. An *neNCE grammar* is a system $G = (\Sigma, \Pi, \Delta, \Gamma, \Omega, \mathcal{P}, Z)$, where

(1) Σ is an alphabet of *node labels*,
(2) Π ($\subseteq \Sigma$) is the set of *terminal node labels* (the elements in $\Sigma - \Pi$ are *nonterminal node labels*),
(3) Δ ($\subseteq \Pi$) is the set of *final node labels* (the elements in $\Pi - \Delta$ are *nonfinal node labels*),
(4) Γ is an alphabet of *edge labels*,