Graph grammars

Jiří Zuzaňák

Brno University of Technology

December 19, 2007

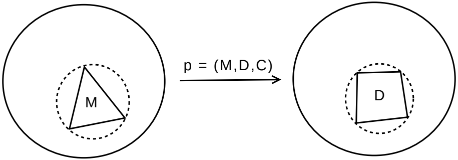
Introduction

- Graph grammars originated in the late 60s.
- Motivated mainly by problems of pattern recognition, compiler construction and program optimization.
- Area of graph grammars generalizes theory of formal languages based on strings and the theory of term rewriting based on trees.
  - Chomsky grammars $\longrightarrow$ graph grammars
  - text rewriting $\longrightarrow$ graph transformation
  - textual description $\longrightarrow$ visual representation
- This theory has wide range of practical applications.

Graph grammar approaches

- Node replacement (Rozenberg, Engelfried)
- Hyperedge replacement (Habel, Kreowski)
- Algebraic Approaches
  - Double Pushout (Ehrig, Schnider, Corradini et al)
  - Single Pushout (Raoult, Löwe et al)
  - Pullback (Bouderon)
  - Double Pullback (Heckel et al)
- Logical approach (Courcelle, Bouderon)
- 2-Structures (Rozenberg)
- Programmed graph replacement (Schuerr)

Graph grammar



- A *graph grammar* is a pair $G = (S, P)$, where S is a starting graph and P is a set of production rules.
- *Graph replacement grammars* have production of form (M, D, C), where M is the mother graph, D is daughter graph, and C is collection of connection instructions.
- All occurrences of graph M in a host graph are replaced with D using the set of connections C.

Node Label Controlled (NLC) Graph Grammars

Definition

A *node label controlled graph grammar* is a 5-tuple $G = (\Sigma, \Delta, P, C, S)$, where

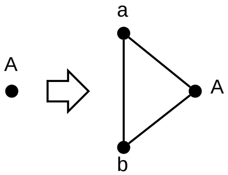
- Σ - is alphabet of node labels
- Δ - is alphabet of terminal node labels, $\Delta \subseteq \Sigma$
- P - is finite set of productions which are pairs (d, Y), with $d \in \Sigma$, and Y is graph.
- C - is connection relation, a function from Σ to 2^{Σ}
- S - is initial graph

Productions replaces single vertex by graph. Connection relations are specified globally for all productions. Connection relations apply only on neighboring vertices to mother graph (vertex) M.

Node Label Controlled (NLC) Graph Grammars

Example

Let $\Delta = \{a, b\}$, $\Sigma = \{A, a, b\}$, and S be a node with label A, then bellow are two productions from P and connection relation C.



$$C = \left\{ \begin{array}{cc} A, & \{a, b\} \\ a, & b \\ b, & a \end{array} \right\}$$

Neighborhood Controlled Embedding (NCE) Graph Grammar

Definition

The *neighborhood controlled embeding* (NCE) graph grammar is a 4-tuple $G = (\Sigma, \Delta, P, S)$, where Σ, Δ and S are defined as in NLC graph grammar.

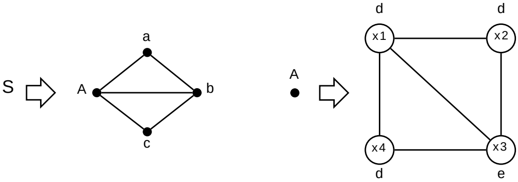
- P is finite set of productions, which are 3-tuples (d, Y, C), with $d \in \Sigma$ and Y is graph.
- C is connection relation, $C \subseteq \Sigma \times V_Y$, and V_Y is a set of nodes of Y.

Set of graphs generated by NCE grammars is same as set of graphs generated by NLC graph grammars.

Neighborhood Controlled Embedding (NCE) Graph Grammar

Example

Example of initial graph S and a production with connection relation of NCE grammar are given below.



$$C = \left\{ \begin{array}{cc} a, & x1 \\ b, & \{x2, x3\} \\ c, & x4 \end{array} \right\}$$

Edge-Labeled Directed Neighborhood Controlled Embedding (edNCE) Graph Grammar

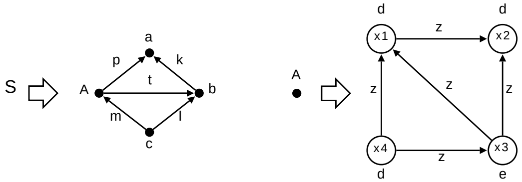
An edge-labeled directed neighborhood controlled embedding (edNCE) graph grammar is 7-tuple $G = (\Sigma, \Delta, \Gamma, \Omega, P, C, S)$, where

- Σ - is an alphabet of connection labels.
- Δ - is an alphabet of terminal node labels, $\Delta \subseteq \Sigma$.
- Γ - is an alphabet of edge labels.
- Ω - is an alphabet of terminal edge labels, $\Omega \subseteq \Gamma$.
- P - is a finite set of productions of the form (d, Y, C), with $d \in \Sigma$ and Y is graph.
- C - is a connection relation, $C \subseteq \Sigma \times \Gamma \times \Gamma \times V_Y \times \{in, out\}$.

Edge-Labeled Directed Neighborhood Controlled Embedding (edNCE) Graph Grammar

Example

First line in connection records list means that vertex $x1$ will be adjacent to vertex a, if vertex a is adjacent to mother vertex A on the edge labeled p. Label of the edge between $x1$ and a will be d, and direction of this edge will be from $x1$ to a.



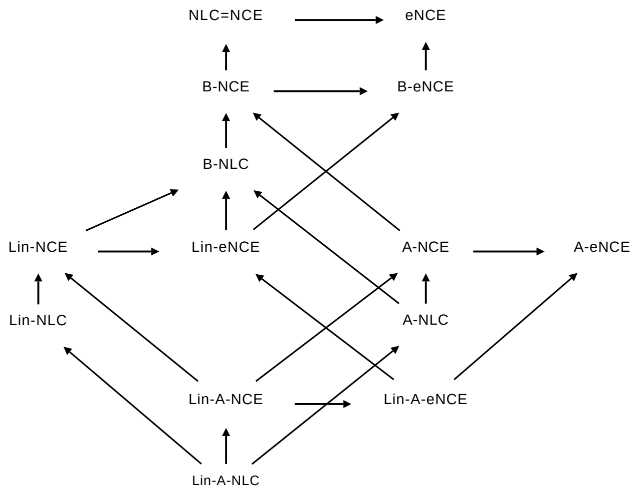
$$C = \left\{ \begin{array}{ccccc} a, & p, & d, & x1, & in \\ b, & t, & f, & x2, & in \\ c, & m, & g, & x3, & in \end{array} \right\}$$

Restriction on connect relations

- (B) *Boundary graph grammars* - no two non-terminals are adjacent in right hand side of each production, and in start graph.
- (Lin) *Linear graph grammars* - at each derivation step daughter graph contain at most one non-terminal.
- (A) *Apex graph grammar* - Connection instruction contains only terminal nodes.
- (-) *Regular graph grammars* - The right hand side is a single non-terminal or consist of connected terminal and nonterminal.

The desired property of new formed graph grammar classes is *confluence*. Grammar is *confluent*, if the result of derivation does not depend on the order of derivations.

Node replacement graph grammars hierarchy



Edge replacement graph grammar

Definition

An *edge replacement graph grammar* is a 7-tuple $G = (\Sigma, \Delta, \Gamma, \Omega, P, C, S)$, where Σ, Δ and S are defined as before, and

- Γ is alphabet of edge labels.
- Ω is alphabet of terminal edge labels, $\Omega \subseteq \Gamma$
- P is a finite set of productions of the form (e, Y, C), where e is a single label of edge from $(\Gamma \backslash \Omega)$, Y is a graph.
- C is a gluing relation $\left\{ \begin{array}{c} head(e) \to begin(Y) \\ tail(e) \to end(Y) \end{array} \right\}$

Edge replacement graph grammar

- In every production, the graph Y has two nodes marked *begin* and *end*.
- When a production is applied, edge e is removed from host graph, and vertices incident to e, are replaced by vertices marked *start* and *end*.
- Vertices of host graph previously incident to edge e preserve all other connections to the host graph.
- Labels of this two vertices are replaced by labels in graph Y.

Edge replacement context sensitive graph grammar

Definition

Edge replacement context sensitive graph grammar is a 7-tuple $G = (\Sigma, \Delta, \Gamma, \Omega, P, C, S)$, where

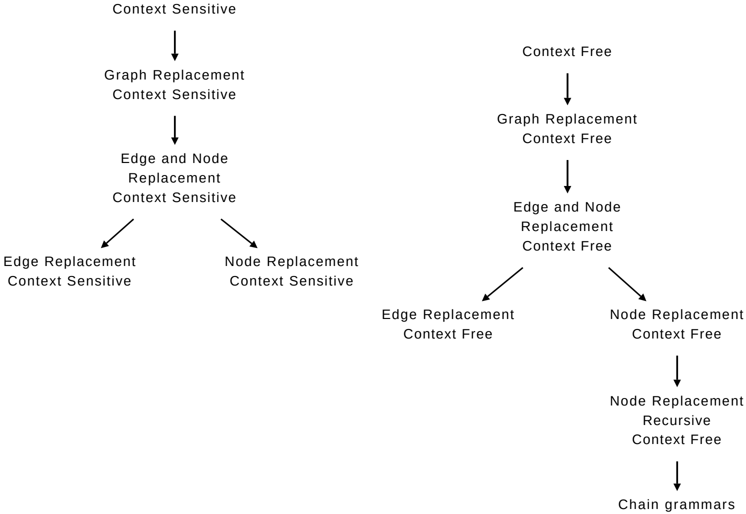
- Σ, Δ, Γ, Ω, and S are defined as before.
- P is a finite set of conditional productions of the form

$$if \left\{ \begin{array}{c} neighborhood(head(e)) \subseteq O \text{ and} \\ neighborhood(tail(e)) \subseteq P \end{array} \right\} than\ e \rightarrow Y \text{ and } C = \left\{ \begin{array}{ccc} head(e) & \rightarrow & begin(Y) \\ tail(e) & \rightarrow & end(Y) \end{array} \right\}$$

  where, e is a single edge, with a label from $(\Gamma \backslash \Omega)$ and Y is a graph.
- C is gluing relation, where O and P are sets of labels, $O, P \subseteq \Sigma$.

Replacement of edge e by graph Y occurs only if a specified conditions are met.

Graph grammars hierarchy



Double Pushout

Definition

Rewrite rule is defined as a pair of morphisms $L \leftarrow K \rightarrow R$, where L is *left hand side* of rule, R is *right hand side* of rule and K is the *interface graph*.

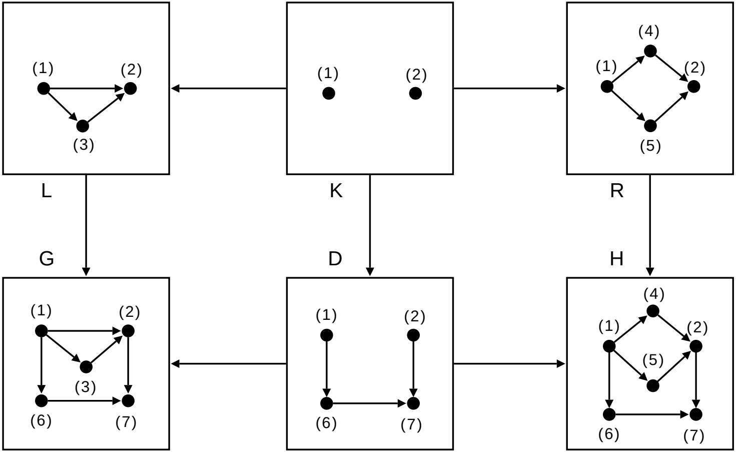
Occurrence of this rule in graph G is morphism of L to G. Such occurrence is rewritten by constructing the diagram from bottom.

$$\begin{array}{ccccc} L & \leftarrow & K & \rightarrow & R \\ \downarrow & & \downarrow & & \downarrow \\ G & \leftarrow & D & \rightarrow & H \end{array}$$

The part of L outside K is deleted from G, and replaced by the part of R outside K. Thus creating result graph H.

Double Pushout

Example



Single Pushout

Definition

Rewriting rule is defined as a partial graph morphism $L \Rightarrow R$, where L is left hand side of rule and R is right hand side of rule.

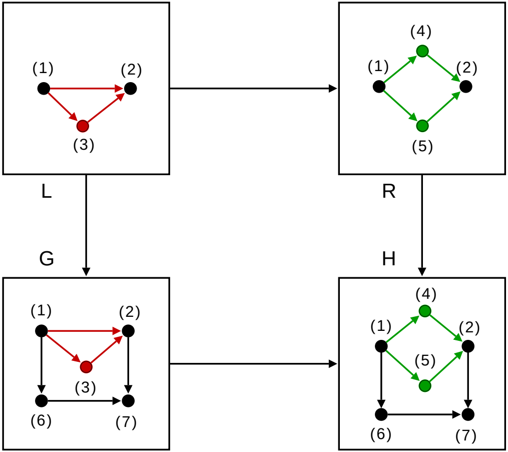
- Partial morphism of A to B is morphism of subobject A to B: $A \hookleftarrow X \rightarrow B$.
- Graph rewriting by single pushout is described at bottom diagram.

$$\begin{array}{ccc} L & \Rightarrow & R \\ \Downarrow & & \Downarrow \\ G & \Rightarrow & H \end{array}$$

- Conditions for existence single-pushout partial morphism are less restrictive than for double-pushout.

From graph G is removed occurrence of $L \backslash R$, and then added copy of $R \backslash L$.

Single Pushout



Application of graph grammars

- program optimization, flow graph representation and modification
- pattern recognition (image processing, music recognition)
- data network connection optimization
- functional and logic programing languages
- data mining mechanisms optimization
- neural networks with variable architecture
- ...

Bibliography

J. Cuny, H. Ehrig, G. Engels, G. Rozenberg:
Graph grammars and their application in computer science.
Berlin: Springer Verlag, 1996