Hyperedge Replacement Languages and Pushdown Automata

SILVIU DUMITRESCU
Department of Informatics
University Transilvania of Brasov
Iuliu Maniu 50
ROMANIA

Abstract: - In this paper we are studying the relations between generated and accepted hyperedge replacement languages. In context freeness of hyperedge replacement grammars we can transform each grammar into an equivalent one in Greibach Normal Form (HRGNF). In order to create a pushdown automata for hyperedge replacement languages (PDAH) we build an algorithm to transform the planar structure of a hypergraph into a linear list of labels and finally into a linear list of hypergraphs. With this kind of lists we can build a PDAH from existing HRGNF. For each grammar exists a PDAH which recognize the same language.

Key-Words: - Hyperedge Replacement Grammars, Parsing Hypergraphs, Greibach, Context Freeness, Nondeterministic, Pushdown Automata

1 Introduction
It is already well known that graphs, diagrams or images are a way of representing information very spread but with a lot of problems in generating, recognizing or compacting. That’s why a study in domain of graphs and formalizations of graphs could be very interesting [9].

The notion of hypergraph was introduced as a generalized graph and consists of a number of hyperedges connected or not [1]. A hyperedge is an atomic item, generalized from edge, defined by a label and a fixed number of tentacles. On each tentacle is attached a node. Some nodes are external and are involved in hyperedge replacement. We can define grammars which involve replacement and with grammars we can generate languages.

There are different kinds of grammars with set of labels divided in terminals and nonterminals or with only one set of labels [2]. In the second case the set of nonterminals is empty and terminal hyperedges are not labeled. This grammars could be maximum parallel such Lindenmayer systems [7]. The languages generated by such grammars include visual structures like fractals.

In this paper all the grammars considered are context free. So, it does not matter how we choose the starting hyperedge in the replacement and it is not relevant how many times we repeat the replacement, but it’s important to have, in each step of the derivation, a production where the label of the replaced hyperedge exists on its left side. In the second section of this paper we summarize the basic notions involved in hyperedge replacement and some considerations about context freeness and hyperedge grammars.

In the third section of this paper, the main one, we consider a context free hyperedge replacement grammar in Greibach normal form. We build an algorithm which parses all the hypergraphs involved in productions set. Input of the algorithm is the source of the hypergraph. Output of the algorithm is a list of labels. We parse all the nodes starting with that one marked with 1. Each node could be adjacent to many hyperedges. All the adjacent hyperedges could introduce some new unvisited nodes which are enqueued. The algorithm works too for not connected hypergraphs. In the end of the paper we introduce the PDAH, inspired from string pushdown automata. The main differences are input alphabet and stack alphabet which don’t contain letters but hypergraphs. For every HRGNF we can construct a PDAH to recognize the language generated by the grammar.

2 Problem Formulations
We start this section with basic definitions involved in this paper.

2.1 Definitions and notations
Definition 1: [4] A hypergraph, \( H \), is a tuple \((V_H, E_H, \text{att}_H, \text{lab}_H, \text{ext}_H)\) where \( V_H \) is the finite set of
nodes, EH is the finite set of hyperedges, attH: EH → VH* is the application of attaching, which assigns a sequence of pair wise distinct nodes to every hyperedge, labH: EH → C is the application of labeling, which assigns a label to every hyperedge from arbitrary but fixed and not empty set C, and extH ∈ VH is a sequence of pair wise distinct external nodes.

Definition 2: [4] The type of a hyperedge, e ∈ E, is the application type: C → N with \( |\text{type}(\text{lab}(e))| = |\text{att}(e)| \).

For a hypergraph, H, the type of H, type(H), is the number of external nodes, type(H) = |extH|.

Let H = (VH, EH, attH, labH, extH) be a hypergraph and R be a hypergraph over the same set of labels as H is. By H[e | R] we understand the hypergraph obtained from H by replacing hyperedge e from H and adding the hypergraph R so that the i-th external node of R is glued over the i-th attached node of e with i=1,type(e). Moreover, extH[ext[e | R]] = extH,

type(H[e | R]) = type(H).

Definition 3: [4] A hyperedge replacement grammar, HRG, is a tuple (N, T, P, S), where N is the set of nonterminal labels, T is the set of terminal labels, N ∩ T = ∅, P is the set of productions, P = {(A, R) | A ∈ N, R is a hypergraph labeled in N ∪ T}, with \( \text{type}(A^*) = \text{type}(R) \), and S ∈ N is the starting symbol.

A direct derivation in HRG, using productions from P, \( H ⇒ H' \), takes place if and only if exists \( e ∈ EH \) such as (labH(e), R) ∈ P and \( H^* = H[e | R] \).

The language generated by the hyperedge replacement grammar HRG is \( L(HRG) = \{H \mid \exists S^* ⇒_P H \} \), labH(e) ∈ T ∀ e ∈ EH.

2.2 Context Freeness

In this paper all the hyperedge replacement grammars considered are context free with any other specifications. In the context freeness we underline two major results obtained in [3] and [6].

The first one, for every hyperedge replacement grammar, \( HRG = (N, T, P, S) \), without rewritings and \( λ \)-free, exists an equivalent grammar, \( HRCNF = (N_1, T, P_1, S) \), in Chomsky Normal Form. That means all productions in P1 are by the form (A, H), where \( A ∈ N_1 \) and \( |E_H| = 1 \), lab(e) ∈ T, e ∈ EH or \( |E_H| = 2 \), lab(e) ∈ N, e ∈ EH, i = 1,2.

The second one, for every hyperedge replacement grammar in Chomsky Normal Form, \( HRCNF = (N_1, T, P_1, S) \), exists an equivalent grammar, HRGNF = (N2, T, P2, S), in Greibach Normal Form. That means all productions in P2 are by the form (A, H), where \( A ∈ N_2 \) and labH: \( E_H → T ∪ N_2 \), |E_H| ≥ 1 with exactly one hyperedge labeled in T.

The hierarchies of nonterminal and production sets are: \( N ⊆ N_1 ⊆ N_2 \), P ⊆ P1 ⊆ P2.

3 Pushdown Automata for Hyperedge Replacement Languages

In string grammars we have pushdown automata [8] as counterpart of context free string grammars [10].

Inspired by this device we introduce in this paper an automaton to analyze a hyperedge replacement language. We proof that for each hyperedge replacement grammar we can construct a nondeterministic pushdown automaton. After that we proof that the language accepted by pushdown automata and the language generated by hyperedge replacement grammar are equivalent.

First we transform the planar structure of the hypergraph into a linear one, of labels. For this we use an algorithm which scans the hypergraph. It starts with the source hyperedge and using the list of attached nodes continues with adjacent unvisited hyperedges. Every time when we find an unvisited hyperedge we add its label to a list of visited labels. The algorithm stops when all hyperedges were visited.

Let H = (VH, EH, attH, labH, extH) be a hypergraph. We consider each attached node marked with i, where i=1, |VH|, \( |V_H| \). We use next algorithm to obtain the linear parsing of its labels, \( \text{LIST}_H \).

1. \( \text{LINEAR}(H, \text{LIST}_H) \)
2. \( \text{LIST}_H ← ∅ \)
3. for each hyperedge \( e ∈ E_H \) do
4. \( \text{visit}(e) ← \text{false} \)
5. endfor
6. \( Q ← ∅ \)

A rewriting is defined in [3].

---

1) For a set V, \( V^* \) denotes the set of all strings over V, including the empty string λ; \( V^+ = V^- \{\lambda\} \) denotes the set of all strings over A except the empty string λ.
2) For w ∈ V*, |w| denotes the length of w.
3) A λ-production is defined in [3].
4) \( A^* \) represents a hypergraph with one hyperedge labeled with A.
5) \( λ ⇒_P^* \) represents a derivation in k steps, k ≥ 0, with productions from P (reflexive and transitive closure)
source ← first(E_H)
8. visit[source] ← true
9. add (LIST_H, lab(source))
10. for each node x ∈ attlab(source) * do
11.    ENQUEUE(Q, x)
12.  V_H ← V_H(x)
13. endfor
14. while Q and V_H are not empty do
15.   if Q is empty
16.      source ← next_unvisited(E_H)
17.      visit[source] ← true
18.      add (LIST_H, lab(source))
19.   for each node x ∈ attlab(source) * do
20.      ENQUEUE(Q, x)
21.      V_H ← V_H(x)
22. endfor
23.   endif
24.   x ← head(Q)
25.   DEQUEUE(Q)
26. for each hyperedge e ∈ E_H do
27.   if not visit[e] and x ∈ E_H(e) *
28.      add (LIST_H, lab(e))
29.   for each node y ∈ attlab(e) * do
30.      if y ∈ V_H
31.        ENQUEUE(Q, y)
32.        V_H ← V_H(y)
33.     endif
34.   endif
35. endfor
36. endfor
37. endwhile

Procedure LINEAR works as follows. At lines 3-4 assigns the value false, in vector visit, to every hyperedge. Q is the queue where are stored parsed nodes. Initial Q is empty (line 6). Variable source denote the entrance of the hypergraph. We can consider the entrance of the hypergraph one of the hyperedges adjacent with the node marked with 1 (line 7). Corresponding to this it adjusts the value of visit with true (line 8), adds the label to the list, LIST_H (line 9), inserts all attached nodes in Q and removes from VH. The operation of enqueuing and dequeuing takes O(1) time. Because a node is only once enqueued and dequeued the total time devoted to queue operations is O(|V_H|). Thus, the total running time of LINEAR is O(|E_H|⋅|V_H|) which means square time complexity.

With previous procedure we can transform into a linear structure every hypergraph even it’s connected or not.

Definition 3: A nondeterministic pushdown automaton for hyperedge replacement languages, PDAH, is a system (Q, Σ, Γ, δ, q_0, Z_0), where:
- Q is a finite set of states;
- Σ is an alphabet of hypergraphs, called the input alphabet
- Γ is an alphabet of hypergraphs, called the stack alphabet
- q_0 ∈ Q is the initial state
- Z_0 ∈ Γ is the start symbol
- δ: Q×(Σ∪{ε})×Γ → 2^(Q×Γ*) is the transition function.

Instantaneous description of a PDAH is a triple (q, w, γ), where q ∈ Q, w is a string of input symbols and γ is a string of stack symbols.

A transition of PDAH from (q, w, Z_0) to (p, w’, βα), denoted by (q, aw, Z_0) ├ (p, w, βα), exists if and only if (p, β) ∈ δ(q, a, Z), where a may be ε or an input symbol.

We define the language accepted by empty stack of PDAH to be:
{w | (q_0, w, Z_0) ├ (p, ε, ε), p ∈ Q}

Let now HRGNF = (N, T, P, S), be a hyperedge replacement grammar in Greibach normal form. We transform derivations, which use productions of P, into transitions, which use δ function of PDAH.

The first step is to transform all hypergraphs involved in P into linear lists of labels using procedure LINEAR.

After that, we define PDAH as (|q|, T, N, δ, q, S), where (q, γ) ∈ δ(q, a, A) if and only if (A, R) ∈ P, the result of linearization of R using LINEAR is a
string of labels and its transformed into a string of hypergraphs is \( \gamma \).

For instance, let’s say that the list of labels, as result of the algorithm, is abcd. Then with these labels we form the string of hypergraphs \( a \cdot b \cdot c \cdot d \).

Theorem 1: The language generated by HRGNF is accepted by PDAH.

Proof: Classical induction on the length of the derivation in GNF.

4 Conclusions

Context free hyperedge replacement grammars have a behavior very much like context-free Chomsky grammars. Till now we found a lot of common characteristics like normal forms or closer properties [5]. Our next step in proving similar behavior is to create pushdown automata which recognize these languages. In this paper we found one way relation, from the existing HRGNF to PDAH. The reverse problem, to create a HRGNF from existing PDAH is still open.

References: