An efficient implementation of graph grammars based on the RETE matching algorithm

H. Bunke, T. Glauser, T.-H. Tran
University of Bern,
Länggassstr. 51, CH-3012 Bern, Switzerland

ABSTRACT: This paper is concerned with the efficient determination of the set of productions of a graph grammar that are applicable in one rewriting step. We propose a new algorithm that is a generalization of a similar algorithm originally developed for forward chaining production systems. The time complexity of the proposed method is not better than that of a naive solution, in the worst case. In the best case, however, a significant speedup can be achieved. Some experiments supporting the results of a theoretical complexity analysis are described.

Keywords: graph grammars, forward chaining, conflict set, RETE-matching algorithm, computational complexity

CONTENTS

0. Introduction
1. Informal Overview and Example
2. The RETE-network
3. Compilation of the RETE-network
4. Dynamic Behaviour of the RETE-network
5. Computational Complexity Analysis
6. Experimental Examples
7. Summary and Conclusions

0 Introduction

Graph grammars have been a subject of research for about twenty years now and significant progress in both theory and applications can be observed over the past years [1, 2, 3]. A problem that is inherent to graph grammars, however, is the high computational complexity of many operations as compared to string grammars. In this paper, we consider the problem of running graph grammars in a forward directed fashion. That is, we start with the initial graph of a grammar and successively apply productions until a final graph has been derived. The basic algorithm for this task is shown in Fig. 1. The conflict set is determined by checking each production if its left-hand side occurs as a subgraph in the actual graph $g$. If the left-hand side of a production is contained in $g$ in a number of different ways, we say there are several instances, or occurrences, of $p$ in $g$. In the second step of the body of the loop in Fig. 1 we select one instance $p$ of an applicable production. This selection can be made at random or according to some predefined rules, for example, the control diagram of a programmed graph grammar [4]. Finally, in the third step we apply the production that has been chosen in step 2. This is done by deleting the left-hand side of $p$ in $g$, inserting the right-hand side of $p$ into $g$, and applying the embedding transformation procedure in order to ensure the proper connections between the newly inserted right-hand side and that part of $g$ that remains unchanged.
while termination condition is not true do
  determine the conflict set (i.e., the set of applicable productions);
  select one applicable production $p$ from the conflict set;
  apply $p$;

Figure 1: Basic algorithm for running a graph grammar in a forward directed way

after deleting the left-hand side of $p$. After a particular production has been applied, the derivation
is continued, i.e. the body of the loop is executed again, until a termination condition becomes true.
The derivation is usually stopped if a graph with only terminal labels has been derived or if there
isn't any production applicable to the actual graph.

A naive implementation of the forward derivation of a graph grammar that exactly follows the
algorithm shown in Fig. 1 is conceptually fairly simple but computationally expensive, particularly
in determining the conflict set. If we want to determine the conflict set in a straightforward way,
we go in a loop over all productions. For each production $p$ we check the actual graph $g$ in order
to find all occurrences of $p$ in $g$. This is done by taking one after the other node and comparing it
to the nodes of the actual graph. As a result, the naive determination of the conflict set requires of
the order $O(P \cdot G^L)$ operations for an actual graph of size $G$ and a grammar with $P$ productions
where each left-hand side is of size $L$. Note that the factor $P$ comes from the outer loop over all
productions while $G^L$ is the time complexity for finding all occurrences of a particular production
in the actual graph.

In this paper, we present a new method for the efficient determination of the set of all applicable
productions in one rewriting step of a graph grammar. As it will be discussed later, the application
of this method can lead to a significant speed-up of the execution of productions. Our new algorithm
is mainly based on the observation that only local changes occur in the actual graph if a production
is applied. In other words, most of the underlying graph will remain unaffected by a production
application. It follows that a number of productions that have been found applicable in the last
cycle are still applicable in the actual cycle (see Fig. 1). So in the course of a derivation sequence
there are only incremental changes to the conflict set. For the determination of these incremental
changes it is sufficient to do a local analysis that is based on the left- and right-hand side of the
production actually applied. So it is not necessary to inspect all grammar productions and the
complete underlying graph.

The algorithm shown in Fig. 1 is not only applicable to graph grammars but also to forward
chaining production systems. A forward chaining production system consists of a database, which
is the counterpart to the underlying graph in a graph grammar, a finite set of productions, and
an interpreter that follows the algorithm in Fig. 1 in order to derive conclusions from an initial
database. For more details about production systems see [5]. A particular example of a forward
chaining production system is the OPS5 software tool [5]. A production system differs from a
graph grammar mainly in the data structures, which are string-oriented rather than being graphs.
Nevertheless, the determination of the conflict set in a forward chaining production system is a
problem of exponential time complexity [6]. This is caused by the fact that there may be variables
in the productions that require consistent bindings. An algorithm for the efficient determination of
the conflict set in forward chaining production systems has been proposed in [6]. In fact, the RETE
algorithm according to [6] has been incorporated in the interpreter of the OPS5 system [5].
The algorithm presented in this paper is a generalization of the RETE algorithm to graph grammars. As
it will be seen in the following sections, most of the concepts can be transferred from the original
algorithm, i.e. from string oriented structures, to graphs in a straightforward way.

There are applications of graph grammars where it is not necessary to determine the complete
conflict set in each rewriting step. Instead, it could be sufficient to find any occurrence of any
applicable production. In such a situation the selection of an applicable production could be based on