A REDUCTION OF LR PARSING TABLES FOR ARITHMETICAL EXPRESSIONS

Zbyněk KŘIVKA, Master Degree Programme (4)
Dept. of Intelligent Systems, FIT, BUT
E-mail: xkrivk01@stud.fit.vutbr.cz

Supervised by: Dr. Alexandr Meduna

ABSTRACT

The present paper introduces a method that reduces the number of rows and columns in LR parsing tables in terms of context-free grammars for arithmetical expressions. It makes use of common binary operators that have the same priority. This analysis and reduction of LR table is based on a new concept–grammatical tree generated from productions of the grammar under investigation.

1 INTRODUCTION

This paper is expressed in terms of LR syntax analysis, however most of presented approaches can be adapted for most other bottom–up syntax-–analysis methods. Project specification is concentration on LR(0) grammars for arithmetical expressions. The definition and appropriate classification of LR grammars is included in [1].

1.1 GRAMMAR FOR ARITHMETICAL EXPRESSIONS

Consider the grammar $\Gamma = (\ \{E, T, F\},\ \{+, -, *, /, (,), id\},\ P,\ E)$, where

$P = \{\ E \rightarrow E + T,\ E \rightarrow E - T,\ E \rightarrow T,\ T \rightarrow T * F,\ T \rightarrow T / F,\ T \rightarrow F,\ F \rightarrow (\ E\ ),\ F \rightarrow id\ \}.$

2 GRAMMATICAL TREE

Grammatical tree is a tree built from the grammatical productions. Its nodes are labeled with the left–hand or right–hand sides of productions. Every nonterminal included in a node can generate its grammatical subtree.

2.1 ALGORITHM THAT CREATES GRAMMATICAL TREE

Algorithm 2.1.1: Construction of a grammatical subtree for a nonterminal

Input: Context–free grammar $G = (N, T, P, S)$ and a node labeled by $A \in N$.

Output: Grammatical subtree for nonterminal A.

Method:

1. Take productions from P of the form $A \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$, where $A \in N$.
2. Create edge leading from the root (node A) for all $i \in \langle 1, n \rangle$ into new node labeled by β_i.

Algorithm 2.1.2: Construction of a grammatical tree

Input: Context–free grammar $G = (N, T, P, S)$ without inaccessible nonterminals.

Output: Grammatical tree.

Method:

1. Create set M containing all nonterminals from N.
2. Construct the root of tree labeled by S and remove S from M.
3. Until M is empty, repeat step 4:
4. If $X \in N$ is in any node and $X \in M$, then replace X by its grammatical subtree (algorithm 2.1.1) and remove X from M.

Note: Every terminal symbol occurs only once in the grammatical tree.

Definition 2.1.3: Let S be a symbol in grammatical tree. **The nearest left (right) symbol of S** is:

- the root has no nearest left (right) symbol.
- otherwise, it is either left or right neighbouring symbol.
- if S doesn't have any neighbouring symbol, then it is left (right) nearest symbol of S's parent.

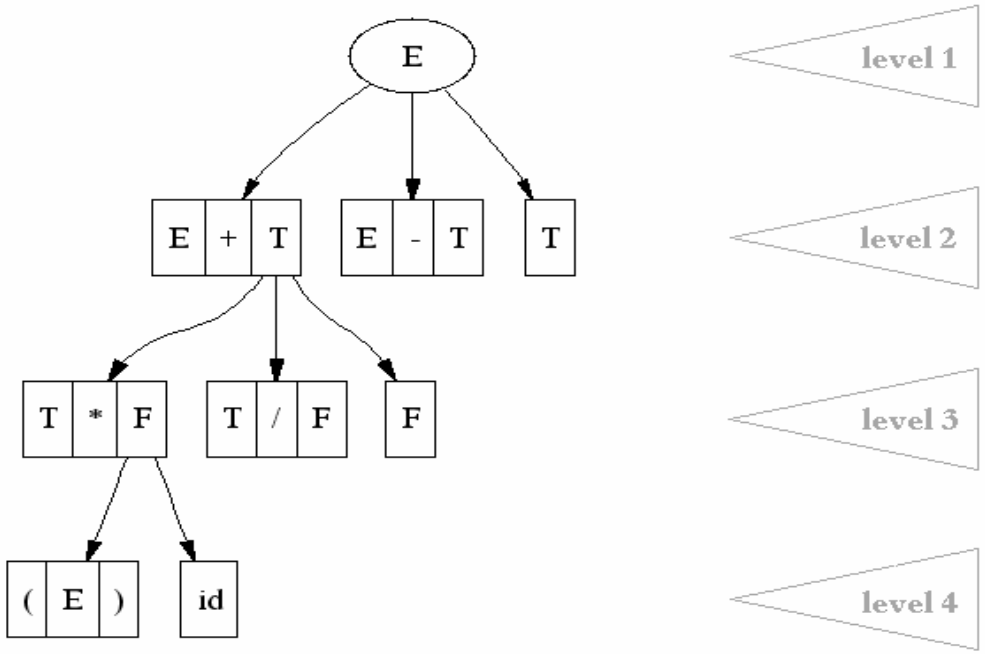

Fig. 1: *Example of grammatical tree for Γ*

2.2 PROPERTIES GAINED FROM GRAMMATICAL TREE

(2.2.1) Terminal *t* is infix **binary operator** if its left and right nearest symbol is nonterminal.

(2.2.2) Binary operators on the same level of grammatical tree structure are of equal **priority**.

3 REDUCTION OF LR PARSING TABLE

(3.1) Transitional edges of LR–state diagram of items can be evaluated by one symbol or set of terminals.

(3.2) Evaluation of LR table column can be also one symbol or **set of terminals**, we write $\{terminal_1, terminal_2, \dots terminal_n\}$.

(3.3) We extend grammatical productions by the possibility of substituting set of terminals for one terminal.

(3.4) An operation equality of terminal and set of terminals corresponds to question: "Is terminal included in this set of terminals?"

Algorithm 3.1: Reduction of LR parsing table

Input: Grammar G = (N, T, P, S) for arithmetical expression and its full–sized LR table (eg. for Γ in [2]).

Output: Reduced LR table

Method:

1. By using (2.2.1) and (2.2.2) create sets of infix binary operators of same priorities $O_1, \dots O_n$ by means of grammatical tree;
   (This step makes sense only for $|O_j| \geq 2$)
2. Create sets of productions $P_1, \dots P_n$ (classified by priority level) where
   $P_j = \{A \rightarrow \alpha o \beta \mid o \in O_j\}$, $j \in \langle 1, n \rangle$. (Note: production $A \rightarrow \alpha o \beta \in P$, $\alpha, \beta \in N$)
3. Choose *k* rows that are distinguished in same columns only by operation reduce rX_i where $\forall\ X_i \in P_j$ and $k = |P_j|$. We join these rows into one (with ignoring ambiguities in some cells with reduction operations).
4. Renumber states (intuitively).
5. Repeat steps 3. and 4. while exist some adepts (rows) for joining
6. For $\forall\ j \in \langle 1, n \rangle$ replace all productions $R_i \in P_j$: $A \rightarrow \alpha o_i \beta$, $i \in \langle 1, k \rangle$, $o_i \in O_j$ by one production $A \rightarrow \alpha O_j \beta$.
7. Renumber new arisen productions and operations of reduction. (By this changing of indexes we achieve disappearance of ambiguities in some cells with reduction.)
8. For $\forall j$: if all columns in action–part of table satisfying terminals $o_i \in O_j$ coincide (except column's evaluation), then join them to a single column evaluated by set O_j.

As you see in step 3, we use comparison of *k* rows and in case of near identity it is possible to group this rows in LR table. We require absolutely identity in goto part and in shift part in operations shift and in blank cells.

Ability of grouping rows together depends on reduction operations for which, there are constructed special sets of reduction productions and these (sets) determine if such a joining is possible.

Due to changing of quantity of states and productions is necessary to renumber their indexes. Just this reindexing causes removing of ambiguity in cells with multi reductions in joined rows.

Last step of algorithm is joining columns corresponding to the terminals in sets of terminals used in new productions.

3.1 EXAMPLE WITH EXPERIMENTAL RESULTS

After proceeding of presented algorithm on full LR table for Γ we obtain reduced LR parsing table showed next:

State/Symbol	**id**	**{+, -}**	**{*, /}**	**(**	**)**	**$**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				accept			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Tab. 1: *Reduced LR table for Γ using sets of terminals*

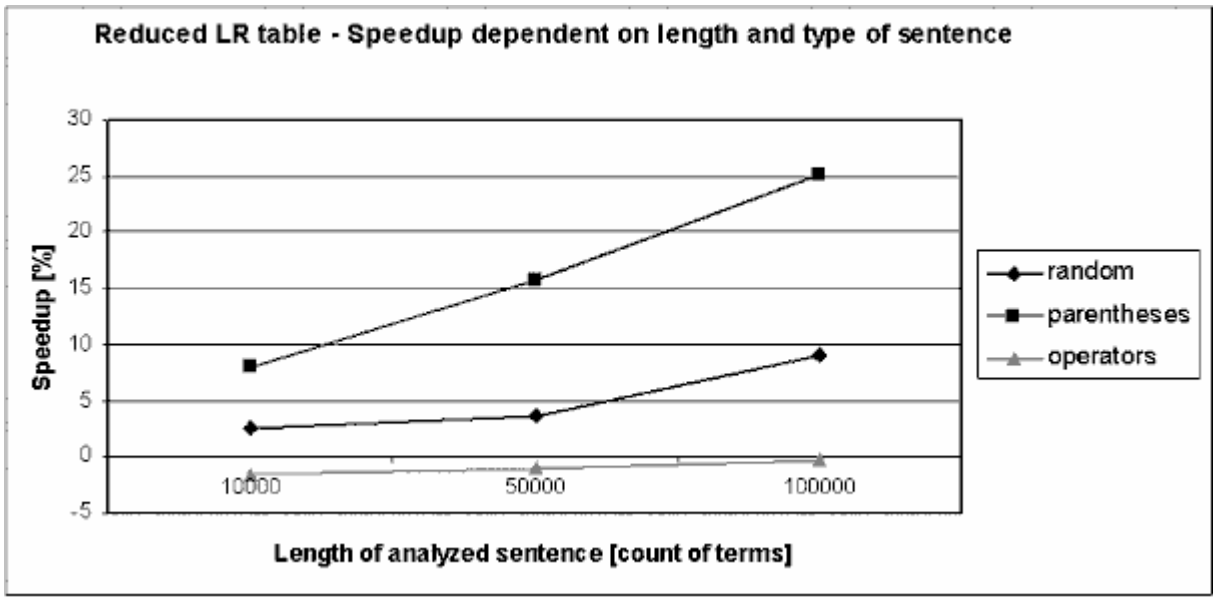

Fig. 2: *Graph of speedup in parsing where reduced LR table for Γ was used*

Note: sentence type "parentheses" = eg. "(((…(id)…)))"; "operators" = eg. "id+id+ … +id".

Observe that if operators make major part of sentence then overhead overtop profit from table reduction. But from the practical point of view, there is a positive speedup around 5% in random sentences.

4 SUMMARY

Usage of this method is constrained to grammars of similar structure, such as Γ and it makes sense when it has a lot of binary operators of same priority.

Advantages:

- significant decreased size of action part of LR table (eg. for Γ up to 50 %).
- analysis needs same quantity of manipulation with stack as with classical method.
- algorithm of LR parsing remains unchanged.

More details and other interesting methods can be found in [3].

REFERENCES

[1] Češka, Milan: Gramatiky a jazyky, VUT FEI Brno, 1992, p. 107-121, (in Czech).

[2] Parsons, Thomas W.: Introduction to Compiler Construction, 1992, p. 132-148.

[3] Křivka, Z.: Taking bottom–up parsing more effective, VUT FIT Brno, 2003, (in Czech).