Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium, Volume 22, No. 1, ISSN 1726-9679
ISBN 978-3-901509-83-4, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2011
*Make Harmony between Technology and Nature, and Your Mind will Fly Free as a Bird*
*Annals & Proceedings of DAAAM International 2011*

# PARSING TABLE STRUCTURE AND ALGORITHM FOR THE LR(K) PARSING METHOD

**ILTSCHEV, V[elko] I[vanov]**

***Abstract:*** *The LR(k)-method uses two tables, which describe the behavior of a push-down authomat, used during the parsing process. These two tables, called action table and goto table, are sparse tables. Moreover, the data in them are not homogeneous in structure since both item numbers and right sides of productions are stored. This paper proposes: a new parsing table structure, which is dense and homogeneous; a parsing algorithm; and an algorithm for generation of this table, based on the SLR(1)-method,.*

***Key words:*** *language processors, compilers, LR parsing method, LR parsing table*

## 1. INTRODUCTION

LR(k)-parsers can be constructed to recognize virtually all programming-language constructs for which context free grammars can be written (Aho et al., 2006). The LR(k) parsing method (Knuth 1965) is a table-driven method that uses a bottom-up strategy. Despite of the recursive-descend method, which uses the program stack, the LR(k)-method manages its own stack. Two tables, called action table and goto table, describe the behavior of a push-down automat, used for parsing. These two tables are sparse tables. Moreover, the data are not homogeneous, since both item numbers and right sides of productions are stored.

This paper proposes: a new parsing table structure, which is dense and homogeneous; a parsing algorithm; and an algorithm, based on the SLR(1)-method, for generation of this table.

## 2. CURRENT PARSING TABLE STRUCTURE

The grammar of arithmetic expressions will be used to explain the method.

$$\left|\begin{array}{l} E \to E+T \\ E \to T \\ T \to T*F \\ T \to F \\ F \to v \\ F \to (E) \end{array}\right. \qquad (1)$$

The graph, which describes the behavior of the push-down automat, used for parsing of this grammar, is shown on Fig.1.

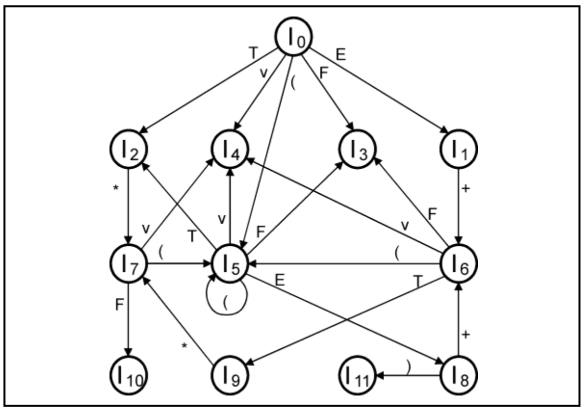
Currently, LR(k)-parsers use two tables to store the graph information. Tab. 1 shows these two tables for the sample grammar above. Because of lack of place, the tables are shown here concatenated.

The disadvantages of this structure are obvios:

- even for such a small grammar (only 6 productions) the tables are very sparse and the larger the number of productions is, the sparser these tables will be;

- additional efforts are necessary to determine, what lies on a crosspoint: a right-hand side of a production, a LR(0)-Item or *accept*.

Modifications over these data structures can be found in (Bonev 2004) and in (Kopp 1988), but they preserve the main idea of two sparse tables with no homogenous structure of data.



.
Fig. 1. Graph of the push-down automat for the grammar of arithmetic expressions

*Action table* *Goto table*

| | v | + | * | ( | ) | ; | E | T | F |
|---|---|---|---|---|---|---|---|---|---|
| $I_0$ | $I_4$ | | | $I_5$ | | | $I_1$ | $I_2$ | $I_3$ |
| $I_1$ | | $I_6$ | | | | **accept** | | | |
| $I_2$ | | T | $I_7$ | | T | T | | | |
| $I_3$ | | F | F | | F | F | | | |
| $I_4$ | | V | V | | V | v | | | |
| $I_5$ | $I_4$ | | | $I_5$ | | | $I_8$ | $I_2$ | $I_3$ |
| $I_6$ | $I_4$ | | | $I_5$ | | | | $I_9$ | $I_3$ |
| $I_7$ | $I_4$ | | | $I_5$ | | | | | $I_{10}$ |
| $I_8$ | | $I_6$ | | | $I_{11}$ | | | | |
| $I_9$ | | E+T | $I_7$ | | E+T | E+T | | | |
| $I_{10}$ | | T*F | T*F | | T*F | T*F | | | |
| $I_{11}$ | | (E) | (E) | | (E) | (E) | | | |

Tab. 1. Action table and Goto table for the push-down automat

## 3. THE PROPOSED TABLE STRUCTURE

The proposed parsing table structure has 4 attributes:

- *CurrentItem* - the LR(0)-Item on the top the of stack

- *NextSymbol* - the next symbol from the input queue

- *Result* - an integer whose meaning depends on the value of the attribute *Action*

- *Action* - if action is ‘S’, then *Result* contains the number of a LR(0)-Item; if action is ‘R’, then *Result* contains the number of a production; if action is ‘A’, then the input queue is recognized as true.

Tab. 2 shows part of the parsing table for the same sample grammar of arithmetic expressions. It is obvious that the structure of data is both dense and homogenous.

## 4. THE PARSING ALGORITHM

The parsing algorithm uses a stack with two sections. The first one contains parts of the input queue and results of reductions. The second one contains the LR(0)-Items the parser has gone throught.

A search function has been written. It receives as parameters: the next symbol from the input queue and the LR(0)-Item on the top of the stack. This function performs a search in the parsing table on attributes *CurrentItem* and *NextSymbol*, and returns the value of the attribute *Action*.

A possible prototype of this function could be:

*char ParsingTable::GetResult(char NextSymbol, int &Result);*

The parameter *Result* is passed by reference. On call, the LR(0)-Item on the top of the stack is passed through this parameter. On return, the value of the table attribute *Result* is returned through this parameter.

The particular steps of the parsing algorithm are:

1. Put into stack: the symbol for bottom of stack and the starting LR(0)-Item (for example $I_0$).

2. Call the search function with the next symbol from the input queue and with the LR(0)-Item on the top of the stack. If the search function returns:

2.1. an ‘S’, then the *Result* parameter contains the number of a LR(0)-Item. In this case put this item, together with the next symbol from the input queue, into the stack and return to step 2.

2.2. an ‘R’, then the *Result* parameter contains the number of production, on which a reduction must be made. In this case pull from the stack the right-hand side of this production, together with the corresponding LR(0)-Items. Perform a second search with the left-hand side of this production and with the LR(0)-Item on the top of the stack. If the search function returns an ‘S’, then go to step 2.1. If the search function returns an ‘E’, then a syntax error has been encountered.

2.3. an ‘A’, then the input queue has been recognized.

2.4. an ‘E’, then a syntax error has been encoutered.

## 5. PARSING TABLE GENERATION

### 5.1 Generating the collection of LR(0)-Items

The SLR(1)-Method, described in (DeRemer 1971) and slightly modified in (Kopp 1988), is used to generate the collections of LR(0)-Items.

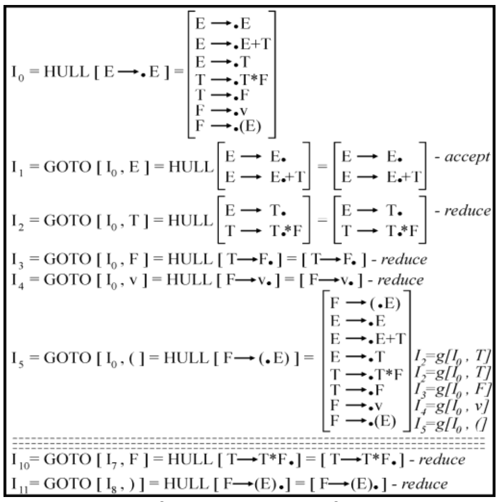
Two functions HULL and GOTO are used for this purpose:

- GOTO describes the transition from one LR(0)-Item, through a symbol, to a new LR(0)-Item. GOTO involves all productions from the current LR(0)-Item where a pass over the same symbol must be made. The GOTO syntax is:

NewItem = GOTO[OldItem, Symbol] (2)

- HULL builds the set of productions, before which the reading head can stay, after a GOTO has been made. The HULL syntax is:

HULL[<set of productions>] (3)



I₀ = HULL [ E ⟶ •E ] = [E ⟶ •E; E ⟶ •E+T; E ⟶ •T; T ⟶ •T*F; T ⟶ •F; F ⟶ •v; F ⟶ •(E)]
I₁ = GOTO [ I₀ , E ] = HULL [E ⟶ E•; E ⟶ E•+T] = [E ⟶ E•; E ⟶ E•+T] - accept
I₂ = GOTO [ I₀ , T ] = HULL [E ⟶ T•; T ⟶ T•*F] = [E ⟶ T•; T ⟶ T•*F] - reduce
I₃ = GOTO [ I₀ , F ] = HULL [ T⟶F• ] = [ T⟶F• ] - reduce
I₄ = GOTO [ I₀ , v ] = HULL [ F⟶v• ] = [ F⟶v• ] - reduce
I₅ = GOTO [ I₀ , ( ] = HULL [ F⟶ (•E) ] = [F ⟶ (•E); E ⟶ •E; E ⟶ •E+T; E ⟶ •T; T ⟶ •T*F; T ⟶ •F; F ⟶ •v; F ⟶ •(E)]
I₂=g[I₀ , T]
I₂=g[I₀ , T]
I₃=g[I₀ , F]
I₄=g[I₀ , v]
I₅=g[I₀ , (]
I₁₀= GOTO [ I₇ , F ] = HULL [ T⟶T*F• ] = [ T⟶T*F• ] - reduce
I₁₁= GOTO [ I₈ , ) ] = HULL [ F⟶(E)• ] = [ F⟶(E)• ] - reduce

Fig. 2. LR(0)-Items for the grammar of arithmetic expressions

The process continues until the set of LR(0)-Items becomes stable, i.e. no new items are generated. A proof that the process of generating new items is convergent is provided in (DeRemer 1971).

Fig. 2. shows the process of generating the collection of LR(0)-Items for the grammar of arithmetic expressions.

The FOLLOW sets for the 3 nonterminal symbols are:

FOLLOW(E) = {+, ), ;}
FOLLOW(T) = {+, *, ), ;} (4)
FOLLOW(F) = {+, *, ), ;}

### 5.2 Filling out the parsing table

The algorithm is:

1. For each GOTO operation put: OldItem into CurrentItem; Symbol into NextSymbol; NewItem into Result and ‘S’ into Action.

2. For each Item, where the reading head has reached the end of a production, a reduction has to be made. In such case, for each symbol, that follows the left-hand side of the production put: the Item into CurrentItem; the FOLLOW-symbol into NextSymbol; the number of production into Result and ‘R’ into Action.

3. For each Item, where the reading head has reached the end of a production, which contains only the starting nonterminal symbol of the grammar, put an ‘A’ instead of ‘R’ into Action.

Tab. 2. shows part the parsing table, generatged in this way.

| CurrentItem | NextSymbol | Result | Action |
|---|---|---|---|
| 0 | v | 4 | S |
| 0 | ( | 5 | S |
| 1 | + | 6 | S |
| 1 | ; | 0 | A |
| 2 | + | 3 | R |
| 11 | ; | 7 | R |

Tab. 2. Part of the new parsing table for the push-down automat

## 6. CONCLUSION

The proposed data structure for the LR(k)-parsing table is both dense and homogenous. This saves memory and simplifies the parsing algorithm, since no additional efforts are necessary to determine the type of action the parser must perform.

Furthermore, the proposed parsing table could be sorted by both search attributes, namely *CurrentItem* and *NextSymbol*. This provides an opportunity for a dichotomic search, resulting in speeding up the search process to $\log_2(N)$.

## 7. REFERENCES

Aho, A.V.; Lam M.S.; Sethi, R. & Ullman, J.D. (2006). *Compilers: Principles, Techniques, and Tools (2nd Edition)*, Addison Wesley, ISBN: 0-321-48681-1, Boston

Bonev, S. (2004). A flexible table driven LR(1) parser, *Proceedings of the 5th international conference on Computer systems and technologies CompSysTech '04*, 17-18 June, 2004, Rousse, Bulgaria, ISBN: 954-9641-38-4, pp. IIIB.12.1 - IIIB.12.6, ACM, New York

DeRemer, F.L.(1971). Simple LR(k) grammars, *Communi-cations of the ACM*, Vol. 14, No. 7, July 1971, page numbers 453 - 460, ISSN: 0001-0782

Knuth, D.E. (1965). On the Translation of Languages from Left to Right, *Information and Control*, Vol. 8, No. 6, December 1965, page numbers 607 - 639, ISSN: 0019-9958

Kopp H. (1988). *Compilerbau: Grundlagen, Methoden, Werkzeuge*, Hanser Fachbuchverlag, ISBN: 3-446-15245-8, Munchen