A Short Description of A Translator Writing System (BOBS - System)

Bent Bruun Kristensen
Ole Lehrmann Madsen
Bent Bæk Jensen
Søren Henrik Eriksen

DAIMI PB - 11
February 1973

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF AARHUS
Ny Munkegade, Bldg. 540
DK-8000 Aarhus C, Denmark

A SHORT DESCRIPTION OF A TRANSLATOR WRITING SYSTEM

(BOBS - SYSTEM)

BY

BENT BRUUN KRISTENSEN
OLE LEHRMANN MADSEN
BENT BÆK JENSEN
SØREN HENRIK ERIKSEN

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MATEMATICS

UNIVERSITY OF AARHUS
DENMARK

INTRODUCTION

This paper is a short description of a translator writing system called the BOBS-SYSTEM. It is an implementation of some of the ideas in the Ph. D. thesis of De Remer [1]. The class of grammars in consideration is the LR(K)-grammars, first described by Knuth in [3], whose implementation requires very large tables. However De Remer claims that by using his techniques one achieves parsers, which are competitive in both space and time with precedence parsers. Horning and Lalonde discuss this topic closer in [5]. De Remer defines a hierarchy of LR(K)-grammars in ascending order of complexity by LR(0), SLR(K), LALR(K), L(M)R(K) and LR(K). What we have done is to implement a parsergenerator in the programming language Pascal [6] for the SLR(1)-grammars.

NOTATION

The reader has to be familiar with finite state machines (FSM's), deterministic push down automata (DPDA's), and context free grammars. An inadequate state is a state in the FSM where applying a production is inconsistent with applying other productions or reading symbols at the same time.

GLOBAL DESIGN OF THE SYSTEM

First of all the system is divided into two parts:

1) the parser-generator

2) the parser

In further detail you can divide the parser-generator in the following parts:

1) input of source grammar
2) grammarchecks
3) generation of the LR(0)-machine

4) Extension to SLR(1) through global look-ahead
5) Optimization according to De Remer
6) Conversion to DPDA (deterministic push-down automata)
7) Further optimizations

INPUT OF SOURCE GRAMMAR

The source grammar has to be written in a slightly modified BNF (Backus Normal Form).

GRAMMARCHECKS

This part can be used as an independent part of the system. If you are designing a grammar for a language it has turned out, that the implemented grammarchecks are very helpful . But of course the grammarchecks are an important part of the system as a whole, because you cannot produce the LR(0)-machine for an ambiguous grammar.

The following is a description of the grammarchecks that the system performs:

1) Left and right recursion.

   The systems checks, whether any nonterminal is both left and right recursive. If so, the grammar is ambigious.

2) Termination.

   The system checks, that all nonterminals can produce a string of only terminals.

3) Erasure.

   The system checks,whether any nonterminal can produce the empty string. If so the grammar is modified so it cannot. (The modified grammar produces the same language).

4) Identical productions.

   The grammar is modified by removing the needless productions.

5) Unused productions.

The system checks, that every nonterminal except the goalsymbol appears in both left and right side of a production.

6) Removing simple productions.

A simple production is a production, of which left and right side consists of only a single nonterminal, and the left side nonterminal does not appear on the left side of any other production. The grammar is modified by eliminating all the simple productions.

7) Connection.

The system checks, that all nonterminals can be derived from the goalsymbol.

GENERATION OF THE LR(0) AND EXTENSION TO SLR(1).

The LR(0)-machine is derived by using the technique developed by De Remer in [1] and [2]. If the LR(0) is generated without any inadequate states, then the source grammar is LR(0). On the contrary, if inadequate states exist, you have to repair the machine by making look-ahead. In our case we have only implemented the global one look-ahead (SLR(1)), and if this is not sufficient one has to change the source grammar.

OPTIMIZATION (5,6 and 7).

If the source grammar happened to be LR(0) or SLR(1), you will get a rather big machine in both space and time.

Two major optimizations suggested by De Remer are therefore performed at this point. Furthermore a more tricky optimization due to Lalonde [4] is done to the machine.

OUTPUT FROM THE PARSER-GENERATOR

1) A list of the source grammar, exactly as you have written it. Any error according to the syntax of input is marked.

2) The results of the earlier mentioned grammarchecks.

3) The grammar written ina nice BNF with possible modifications.

4) Description of states which are not SLR(1).

5) An error message table for use when parsing a string in error.

6) The parsetables (the optimized LR(0) or SLR(1) machine) in the form of a selfcontained Pascal-program.

If there are errors in input you will only get 1.Logical errors in the grammar will give you 1, 2 and 3. If there are no logical errors and the grammar is not SLR(1) you will also get 4. In the case that the grammar is SLR(1) you will get 1, 2, 3, 5 and 6.

In addition there exist several other output facilities, mentioned in [9] but they are only of little interest for this paper.

THE PARSER

As mentioned the parser is a pascal-program, which without changes will check the syntax of an input-string written in the language defined by the source-grammar. If you want to add semantic actions this is possible.

The parser is roughly divided into three parts:

1) Lexical analysis
2) Syntax analysis
3) Error-recovery

LEXICAL ANALYSIS

Because of the important role of an effective lexical analysis for the parser several are designed. Two are to be mentioned here. The first is a fairly general but simple one which only transforms the sourceinput into a string of internal values. The second is of greater importance in our point of wiew, since it collects identifiers, constants and strings (defined in a Pascal-like manner), which makes the tables smaller and the parsing faster.

SYNTAX ANALYSIS

The syntax analyser uses the machine produced by the generator, to parse the input-string. When the syntax analyser makes a reduction a procedure code is called with the number of the production (reduction) as a parameter. The user may then decide, what kind of semantic action, he wants to perform. This is done by writing the body of the procedure 'CODE', which is the only place one has to change the program but of course you may also add new procedures and declarations.

ERROR-RECOVERY

Discovering an error under parsing, an error-recovery algorithm is called, which in a Pascal-like manner makes the error-symbol (with arrows under error-symbols and matching numbers in the margin), and try to recover the error and continue parsing.

CONCLUSIONS

The system has not yet been compared with other systems of the same kind, so you cannot tell whether the system is very good or not.

The largest grammar been treated by the system so far, is the grammar of the language Pascal [6]. The SLR(1) machine for this grammar with more than 250 productions occupies about 7000 60 bit words and the speed of the parser is about the same as the speed of the Pascal-compiler available on the CDC 6400. But our parser does not produce code.

Until now only a few others than the group, who have developed the system have been using it, (first available for other users in december 72). But the experience we already have had by the different types of grammars tells that one without too much work, can modify grammars to become SLR(1) and can change parts of the parser to solve the special needs that one might have.

The work was started as a datalogi 2 project by Mr. Peter Kornerup, whose good ideas and great interest have been of great help to the accomplishment of the project.

FURTHER PROJECTS

As the work has been moving along new projects have arisen. One of the more interesting ones is the problem of defining reasonable methods for making semantics. Several papers in this area have been studied but only a few things seem to be useful in practice. A work by Gorrie [7] appears to be one of the better.

REFERENCES

[1] De Remer, F,L.
"Practical Translation for LR(K) Languages".
PH. D. Thesis, Massachusetts Institute of Technology,
Cambridge, Mass. August 1969.

[2] De Remer, F,L.
"Simple LR(K) Grammars".
CACM p. 453-459. (14,7,1971)

[3] Knuth,D. E.
"On the Translation of Languages from Left to Right".
INF. and CONT. p. 607-639. (oct. 1965).

[4] Lalonde
"An efficient LALR-Parser-Generator".
Tech. Report CSRG-2, University of Toronto.
Toronto, Ontario, 1971.

[5] Horning, J. J.
Lalonde, W. R.
"Empirical Comparison of LR(K) and precedence Parsers".
Tech. Report CSRG-1. University of Toronto.
Toronto, Ontario, 1970.

[6] Wirth, N.
"The Programming Language Pascal"
ACTA informatica 1,35-63(1971).

[7] Gorrie, J. D.
"A Processor Generator System".
Tech, Report CSRG-3.
University of Toronto, Toronto Ontario, 1971.

[8] BOBS
Bent Bæk Jensen
Ole Lehrmann Madsen
Bent Bruun Kristensen
Søren Henrik Eriksen
"Obligatorisk datalogi 2 opgave.
SLR(1)-Parsergenerator og Parser".
DAIMI, Aarhus Universitet.

[9] "BOBS-SYSTEM Bruger Vejledning".
Bent Bæk Jensen
Ole Lehrmann Madsen
Bent Bruun Kristensen
Søren Henrik Eriksen
DAIMI PB nr. 10, December 1972
Aarhus Universitet