Univ. u Novom Sadu
Zb. Rad. Prirod.–Mat. Fak.
Ser. Mat. 24, 1 (1994), 289-298

Review of Research
Faculty of Science
Mathematics Series

APPLICATION OF COMPILER COMPILERS FOR CONSTRUCTING A TRANSLATOR

Dušan Surla, Miloš Racković[1]
Institute of Mathematics, University of Novi Sad
Trg Dositeja Obradovića 4, 21000 Novi Sad, Yugoslavia

Abstract

The work briefly describes compiler compilers COCO-2, LEX and YACC. With the aid of these compiler compilers the methodology was illustrated for constructing a translator for robot-oriented programming languages. A comparative analysis was presented of the technique for realization of the translator using the COCO-2 and LEX-YACC compiler compilers.

AMS Mathematics Subject Classification (1991): 68N20, 70B15.
Key words and phrases: Compiler compilers, robot-oriented programming languages.

1. Introduction

The development of theory of translators constructions has been accompanied by ever increasing appearance of software tools for automatic generation of certain parts of the translators, or complete translators, the so-called compiler compilers. These problems have been extensively treated in the literature [13], and several most known generators of syntactic analyzers have been described recently [14].

[1]The work is supported by Science Fund of Serbia, grant no. 0413 through Institute of Mathematics, Novi Sad

Compiler compilers are most often developed for the automation of the process of generating certain portions of translators. The most known compiler compilers are: ALEX [8], LEX [7], HLP84 [6], LINGUIST [2], MUG [3], GAG [5], YACC [4], COCO [14] and COCO-2 [1].

Starting from the example of constructing a translator from a newly-formed robotic programming language NRJ into the robotic programming language MPRJ that has been realized in the "M. Pupin" Institute in Belgrade, the aim of the present work was to demonstrate the method of translator construction and to carry out a comparative analysis of facilities of the COCO-2 and LEX-YACC systems for constructing this translator. The construction of a translator from NRJ into MPRJ programming language has already been described in detail [9, 15, 10, 11, 12].

2. Short description of compiler compilers COCO-2, LEX and YACC

COCO-2 is a compiler compiler based on the generators ALEX and COCO, which unifies them and eliminates some of their shortcomings. The two most important properties of COCO-2 compiler compiler are the combination of description of lexical structure and syntax, together with the semantics of the chosen language in one input document, as well as the implementation of efficient table-guided method for the top-down analysis which enables powerful semantic calculations in combination with rather effective automatic handling of errors. The input to the compiler generator COCO-2 is a text document, i.e. the appropriate compiler specification written in the high-level language COCOL-2 (COCO Language 2). It describes the lexical structure, syntax and semantics of the chosen source language. For such an input COCO-2 generates the scanner's and parser's codes for the chosen language in the programming language MODULA-2.

LEX is a specific software tool which is used in solving diverse problems appearing in the treatment of the input text. However, its most common application is in automation of the process of translator writing, where LEX is used to generate the lexical analyzer of the source language. The source code of LEX program represents a table of regular expressions and the corresponding to them code portions written in language C. By recognizing an input array of characters given in the table of regular expressions an ap-

propriate action is carried out, i.e. the corresponding portion of the code written in language C is executed. The recognition of regular expressions is realized by implementation of the classical deterministic finite automate which is generated via LEX. LEX system has been constructed with the aim to simplify the interaction with the YACC compiler compiler.

YACC is a general software tool which ensures recognition of the syntactic structure of the source program given in the form of specification describing the source language. Compiler compiler YACC transforms such specification into a function in programming language C, which then analyzes the source program given at the input. This function is called parser, and its task is to call a lower level routine, i.e. the lexical analyzer, which recognizes the corresponding tokens at the input. The tokens are then compared to the specification rules, i.e. with the grammar rules, and when one of these rules is recognized the action is ended. The actions represents the code written in programming language C, and they can return some values, as well as use the values that have been returned by some other actions. The generated mechanism possesses some possibilities for error handling.

3. Generation of the lexical analyzer

The input specifications for LEX and COCO-2 system differ in that in LEX are given regular expressions, whereas in COCO-2 the specifications are given in EBNF. A COCO-2 specification contains only the description of those constructions which have a certain meaning for the latter analysis, while the LEX specifications, unless is required that LEX itself gives certain data at the output, define the regular expressions for all constructions of the source language.

The portions of the file for COCO-2 describing the recognition of the identifiers and non-negative integer numbers are given below.

```
ident <<VAR spix:INTEGER>> =
   letter {letter | digit}
   LEX <<Hash(tokenStart,tokenEnd,spix);>>.

UnsignedInteger <<VAR val:INTEGER>> =
digit {digit} LEXLOCAL <<VAR p:CharPtr;  VAR ok:BOOLEAN;>>
 LEX << val:=0;  p:=tokenStart;  ok:=TRUE;
```

```
WHILE ok DO
 val:=val*10+INTEGER(ORD(p^))-INTEGER(ORD('0'));
 ok:=p<>tokenEnd;  IncAddr(p,1);
END; >>.
```

The part of LEX specification for recognition of the identifiers and non-negative integer numbers should contain the following lines:

```
%{
#include "y.tab.h"
%}
D [0-9] L [a-zA-Z]
 %%
{L}[{L}|{D}]* {Hash(yytext,yyleng,yyval); return (IDENT);}

 {D}+ { int i;  yyval:=0;
for (i=0;i <= yyleng;i++)
   yyval:=yyval*10+yytext[i]-'0';
return (NUMBER); } .
```

In both approaches, for identifier recognition the function Hash is called. In LEX, this is the function written in language C and it should be realized by the user in the part of LEX specification for the user's routines. This function has to recognize the identifier and place it in the symbol table, if it has not been already there, and return the corresponding ordinal number. In contrast to this approach, by the realization through the COCO-2 compiler compiler, the function Hash is built in. The recognition of a non-negative number and calculation of its value is similar in both approaches. In the LEX system, at the end of both actions is returned the corresponding value for representing the corresponding token, and this value is used by YACC compiler compiler for further syntactic analysis. The lexical attributes which are also used by YACC, such as ordinal number of the identifier, number value, and the like, are calculated and returned as the values of the variable yyval. In the COCO-2 system, communication with the syntactic and semantic analyzers is realized by direct quoting the name of the appropriate lexical construction, together with the corresponding attributes.

The work involving the COCO-2 compiler compiler is simpler because the input specification is given by EBNF, which is more clear than the

same record given by regular expressions, and the actions themselves can be represented more clearly, as the lexical attributes are presented as the procedure parameters. With LEX, it is possible to use one lexical value yyval, and if we want to express more lexical attributes, some additional actions have to be included, which substantially complicates representation of the construction itself.

4. Generation of the syntactic analyzer and of the translated program

Syntactic analysis in the syntactic analyzer generated via the YACC and COCO-2 systems is carried out in a completely different way. Within YACC, as is the case with the majority of compiler compilers, is implemented the LALR(1) parser, which carries out the bottom-up syntactic analysis, whereas in the COCO-2 compiler compiler is implemented the LL(1) parser which carries out the top-down syntactic analysis. The mode of implementation of the syntactic analyzer itself requires different descriptions of the source language syntax in the YACC and COCO-2 specifications. With YACC is used a description of the syntax via the left-recursive grammar and the description is given by using the rules of the context-free grammar. The COCO-2 specifications defines the syntax of the source language via the EBNF record in which the right-recursion is more natural.

Below are shown the portions of the COCO-2 specification for translating the basic program structures, as well as the declarations of variables and procedures.

```
Prog = LOCAL <<VAR spix:INTEGER;>>
SEM <<InitDat;>>  "program" ident <<spix>> ";"  Block "."
SEM <<CloseDat;>>.

Block = LOCAL <<VAR val,spix,lab:INTEGER;>>
[ "var"  SEM << InitVar;>>   Variables { Variables } ]
{ "procedure" ident <<spix>> ";"  SEM <<NewProc(spix,lab);>>
  "begin" Statement { ";" Statement } "end" ";"
  SEM <<CloseProc(lab);>> }
"begin" Statement { ";" Statement } "end".
```

```
Variables = LOCAL << VAR spix,ind,tyval,val,val1:INTEGER;
VAR defvar:ARRAY [1..100] OF INTEGER;>>
ident <<spix>>
SEM << ind:=1; NewVar(spix,defvar,ind); val:=0; val1:=0;>>
{ "," ident <<spix>>
SEM << ind:=ind+1; NewVar(spix,defvar,ind);>> }
":" Type <<tyval,val,val1>>
SEM <<SetType(defvar,ind,tyval,val,val1);>> ";".

Type <<VAR tyval,val,val1:INTEGER>> =
( OrdinalType <<tyval>>
| "array" "[" UnsignedInteger <<val>>
          ".." UnsignedInteger <<val1>> "]"
SEM <<CheckInd(val,val1);>>  "of" OrdinalType <<tyval>> ).

OrdinalType <<VAR tyval:INTEGER>> =
  "vector" SEM <<tyval:=7;>>
| "rotmatrix" SEM <<tyval:=6;>>
| "rotation" SEM <<tyval:=5;>>
| "thetai" SEM <<tyval:=4;>>
| "frame" SEM <<tyval:=3;>>
| "integer" SEM <<tyval:=0;>>
| "real" SEM <<tyval:=1;>>
| "boolean" SEM <<tyval:=2;>>.
```

The part of the YACC specification by which is translated the program header, as well as the declaration of variables and procedures, is given by:

```
%{  #include <stdio.h>
typedef struct typ  {  int tyval, val, val1; }  TTYPE;
int lab, ind, defvar[100];
%}
%union { int ival;
TTYPE tval;
}
%token <ival>PROG IDENT TVAR PROC BEG TEND ROT ROMA VEC
             FRA REA TINT THET ARR COUN NUMBER TOF BOO
%type <tval> stype
%%
```

```
prog : { InitDat; } PROG IDENT ';'
       block '.' { CloseDat; } ;

block : dekl proc stat ;

dekl :
     | TVAR { InitVar; } svar ;

svar : svar variables ;

proc :proc proced ;

proced : PROC IDENT ';' { NewProc(yyval,lab); }
         BEG statb TEND ';' { CloseProc(lab); } ;

variables : iden ':' stype ';'
            {SetType(defvar,ind,$3.tyval,$3.val,$3.val1);}
          ;

iden : IDENT { ind = 1; NewVar(yyval,defvar,ind); }
     | iden ',' IDENT {ind++;NewVar(yyval,defvar,ind);}
     ;

stype : ordinaltype { $$.tyval = $1; }
      | ARR '[' NUMBER { $$.val = yyval; }
        COUN NUMBER ']'
        { $$.val1 = yyval; CheckInd($$.val,$$.val1);}
        TOF ordinaltype { $$.tyval = $8; } ;

ordinaltype : VEC { $$ = 7; }
            | ROMA { $$ = 6; }
            | ROT { $$ = 5; }
            | THET { $$ = 4; }
            | FRA { $$ = 3; }
            | TINT { $$ = 0; }
            | REA { $$ = 1; }
            | BOO { $$ = 2; } ;
```

In a similar way we could realize a complete description of the translator

from NRJ into MPRJ programming language by using COCO-2 or YACC compiler compiler. The corresponding procedures which are presented are realized either in C or MODULA-2 language, and their declaration is found in the part for subroutines of YACC specification, or in a separate file which is included in the input file for COCO-2. These procedures perform the translation into the corresponding structure of the target language.

It is evident from the enclosed part of the program that the COCO-2 specification is more clear than the YACC specification. In the COCO-2 specification, the semantic attributes are bound in a simple way as the procedures parameters, whereas in the YACC specification is employed a rather unclear mode by using $$, $1, etc. Also, the need for appearance of more semantic attributes at the same non-terminal symbol additionally complicates the semantics description in the YACC specification. Besides, the auxiliary variables which are necessary for the semantic analysis are in the COCO-2 specification used in a more natural way, as local variables in the frame of the procedure which treats the non-terminal symbol which uses these variables. With the YACC specification, it is often necessary to use the global variables, because, when the action for a certain non-terminal symbol consists of several blocks, it is necessary to introduce the global variable, if it is desired this variable retains its value in all these blocks.

Through the COCO-2 specification one cannot describe all the source languages that can be described via the YACC specification. This implies from the difference between the LL(1) and LALR(1) grammars. However, when constructing a translator from NRJ into MPRJ programming language this does not come to the expression because NRJ language can be described in a simple way using the COCO-2 specification.

5. Conclusion

An analysis is provided of the construction of a translator from NRJ into MPRJ programming language using the COCO-2 and LEX-YACC compiler compilers. In both approaches is obtained a fast one-pass translator which is adaptable to certain alterations of both the source and target language. It appeared that the COCO-2 compiler compiler is more suitable for constructing the translator from NRJ into MPRJ programming language than LEX and YACC, because the use of the COCO-2 specification enables definition of the translator specification in a more natural and elegant manner.

References

[1] Dobler H., Pirklbauer K., COCO-2 - A New Compiler Compiler, SIGPLAN Notices, Vol. 25, (1990).

[2] Farrow R., Generating a Production Compiler from an Attribute Grammar, IEEE Software 1(4):77-93, October, (1984).

[3] Ganzinger H., Giegerich R., Moncke U., Wilhelm R., A Truly Generative Semantics-Directed Compiler Generator, Proceedings of the SIGPLAN Symposium on compiler construction. ACM, June, (1982).

[4] Johnson S. C., YACC - Yet Another Compiler-Compiler, Bell Laboratories, (1975.)

[5] Kastens U., Hutt B., Zimmermann E., GAG - A Practical Compiler Generator, Lecture Notes in Computer Science, Springer, (1982).

[6] Koskimies K., A specification language for one-pass semantic analysis, SIGPLAN Notices 19, 6, 179-189, (1984).

[7] Lesk M. E., LEX - A Lexical Analyzer Generator, Bell Laboratories, (1975).

[8] Mossenbock H., Alex - A Simple and Efficient Scanner Generator, SIGPLAN Notices, Vol. 21, May (1986).

[9] Racković M., Surla D., Construction of a Translator for Robotic Languages with the Aid of Compiler-Compiler Coco-2. Part I, XXXVI Yugoslav Conference ETAN, Kopaonik, (in Serbian), 173-180, (1992).

[10] Racković M., Surla D., Implementation of a Translator for Robotic Languages with the Aid of Compiler-Compiler Coco-2, Bull. Appl. Math., BAM 794/92 (LXI), (1992).

[11] Racković M., Construction of the translator for the robotic programming languages, Proceedings of the Conference on Logic and Computer Science "LIRA '92", Novi Sad, October 29-31, 107-114, (1992).

[12] Racković M., Construction of a translator for robot-oriented programming languages, Master thesis, (in Serbian), Novi Sad, (1993).

[13] Raiha K. J., Bibliography on attribute grammars, SIGPLAN Notices 15, 3, (1980).

[14] Rechenberg P., Mossenbock H., A Compiler Generator for Microcomputers, Prentice Hall, (1989).

[15] Surla D., Racković M., Construction of a Translator for Robotic Languages with the Aid of Compiler-Compiler Coco-2. Part II, XXXVI Yugoslav Conference ETAN, Kopaonik, (in Serbian), 181-187, (1992).

REZIME

PRIMENA KOMPAJLER KOMPAJLERA ZA KONSTRUKCIJU TRANSLATORA

U ovom radu su ukratko opisani kompajler kompajleri COCO-2, LEX i YACC. Pomoću ovih kompajler kompajlera ilustrovana je metodologija konstrukcije translatora za robotski orjentisane programske jezike. Data je uporedna analiza tehnika realizacije translatora upotrebom COCO-2 i LEX-YACC kompajler kompajlera.

Received by the editors October 20, 1993