Univ. u Novom Sadu
Zb. Rad. Prirod.–Mat. Fak.
Ser. Mat. 24, 1 (1994), 299-308

Review of Research
Faculty of Science
Mathematics Series

ADAPTATION OF THE COMPILER COMPILER CONSTRUCTED TRANSLATOR TO CHANGES IN THE TARGET LANGUAGE

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

Ervin Varga
Institute of Computer, Control and Measurement
Trg Dositeja Obradovića 6, 21000 Novi Sad, Yugoslavia

Abstract

A translator for robot-oriented programming languages is constructed with the aid of the compiler compiler COCO-2. This method of constructing of a translator appeared to be very suitable, as the resulting translator can easily be adapted to the changes in both the source and target language. Some necessary modifications of the translator are shown, due to alterations in the target language.

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

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

1. Introduction

Compiler compilers are the software tools for automation of the process of generation of particular parts of a translator. Their use in constructing translators is rather common, and it has been the subject of many articles [8, 3, 9].

These programs enable the translator designer to efficiently define the structure of the source language and the corresponding actions of translation into the target language. Simple alterations of the precisely defined specification of the translator enables the translator to easily adapt to the changes in both the source and target language.

The aim of the present work was to exemplify such changes in the translator which translates the programs from the newly-formed robotic language NRJ, not into the robotic programming language MPRJ but into a hypothetic robotic language HRJ. The intention was to demonstrate the advantage of the method of translator constructing with the aid of the compiler-compiler, and that by simple alterations in the translator specification and changes in semantic actions a complete change in the target language is supported.

2. Short description of the translator form NRJ into MPRJ programming language

The translator from NRJ into MPRJ programming language was constructed using COCO-2 compiler compiler [2], which is one of the most known programming systems of this kind. The input data to the compiler generator COCO-2 is a text document, i.e. the corresponding compiler specification written in the high-level language COCOL-2, in which are described the lexical structure, syntax and semantics of the chosen source language. For these input data, COCO-2 generates the code of the scanner and the parser for the chosen language in the programming language MODULA-2.

The translator itself, together with the mode of its construction and implementation, have been described in detail [4, 10, 5, 6, 7]. The source language NRJ is a programming language formed by narrowing the standard robot-oriented programming language PASRO [1] and adding some commands which are not supported in PASRO. MPRJ is a low-level robotic

programming language which has been realized in the "M.Pupin" Institute Belgrade as a constitutive part of the robotic control system EDUC-NET.

In order to illustrate the way of constructing the translator on which the corresponding alterations to cause changes in the target language will be demonstrated, we also present the parts 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;>>.
```

After the key word SEM are stated the semantic actions which translate the commands of the source language into the corresponding commands of the target language. In order to obtain a more readable document these actions are mainly realized in the frame of the procedures written in MODULA-2 which are found in a separate file. On the example of the declaration of variables and procedures of the source language, the corresponding semantic action is described when the target language is MPRJ programming language.

MPRJ language has no declaration of variables, i.e. all variables are built in, so that in the above specification there is no recording of the commands of the target language. However, for each declared variable of the source program it is necessary to determine the name of the user's built in variable of the target language, which in the target program has the role of the initial variable.

When a variable of the same type is declared, the procedure NewVar is called whenever a variable appears, by which a marker is written that the given variable has been declared, and that it cannot be declared again. Also, by this procedure the array defvar is provided with ordinal numbers of all the variables that are declared in the recognized command of the source program.

By recognizing the type of these variables, i.e. by obtaining the value of the non-terminal symbol Type, the procedure SetType is called by which an appropriate name of the target language variable is booked to each variable from the defvar array. In case of complex variables of the source program, such as additionally defined robotic types, array type, and the like, this procedure prescribes an appropriate number of names of the target language

variables to one variable of the source program.

Procedures are declared by calling NewProc, by which a mark is introduced that the recognized procedure has been defined and that a label is determined marking the beginning of the body of this procedure. Before recording the label itself into the resulting file, i.e. into the target program, the jump command is recorded which enables omitting the procedure body during the execution. By translating all the commands which are within the procedure body this process is terminated by calling the CloseProc procedure, which records the command for returning from the subroutine, and a new label to which is necessary to jump in order to omit the procedure body during the program execution.

3. Short description of the hypothetic language HRJ

HRJ is the target language of the PASRO compiler which has been already described [11, 12, 13]. It is a low-level language, close to machine language, and which is composed of two arrays of instructions:

- static instructions, which are executed independently of the concrete computer-controlled robot, and
- robotic instructions.

Static instructions serve to translate the PASCAL parts of PASRO, whereas the role of robotic instructions is to translate the robotic parts of PASRO programming language.

In order the HRJ language could be executed on a concrete machine, a HRJ interpreter is needed that actually emulates the functioning of a hypothetic computer whose set of instructions is just HRJ language.

The memory of the hypothetic computer is composed of three parts:

- instruction memory,
- static memory, and
- dynamic memory.

Within the instruction memory are found the HRJ instructions generated by the translator.

Static memory is organized according to the stack system. It contains the calculation stack which is used in calculation of complex arithmetic expressions, the control data which are used when returning from the subroutines, the values of parameters which are forwarded when returning from subroutine, as well as the local variables.

Dynamic memory serves to store dynamic data structures.

4. Description of some necessary mofifications of the translator due to alterations in the target language

An alteration in the target language requires the corresponding changes in the translator. In case of constructing the translator using compiler compiler it is possible to utilize significant part of the original translator to obtain a new translator.

In the case of the translator from NRJ language, almost all parts of the input file for COCO-2 for which the translator description is given, can be used. It is only necessary to change the corresponding semantic actions which do the translation, which are called after the key word SEM. In the case the original language and the new language have similar structures, a good part of semantic action of the original translator can be used.

A description of the necessary translator modifications is given for the example of translating the program header and declaration of the procedures and variables. The modified version of the corresponding part of the translator specification has the following form:

```
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, addr:INTEGER;
         VAR defvar, defaddr:ARRAY [1..100] OF INTEGER;>>
ident <<spix>> SEM << ind:=1;
                      NewVar(spix,defvar,ind);
                      val:=0;
                      val1:=0;
                      addr:=6; >>
{ "," ident <<spix>> SEM << ind:=ind+1;
                            NewVar(spix,defvar,ind); >> }
  ":" Type <<tyval,val,val1>>
SEM << SetType(defvar,ind,tyval,val,val1);
       SetAddr(addr,tyval,val,val1,defaddr,ind); >> ";".

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;>>.
```

When translating declarations of the variables of NRJ into HRJ language,

in addition to calculation of the ordinal number of the variable, the mark that the latter has been defined and the corresponding type of that variable, it is also necessary to determine the address of the memory location which is bound to that variable. The variable addr is introduced by which is given the address of the subsequent free memory zone within the static memory. The initial value of this variable is 6, because the first several memory words of the current block are used for different control data.

For determining the address of each variable, a new procedure SetAddr is introduced which sets the addresses of the variables at the corresponding values in dependence of the given type, and stores these addresses in the array defaddr.

When translating procedures, the semantic procedures NewProc and CloseProc are called, which perform the identical action as in the case when MPRJ is the target language, but now the jump command and the command for returning from subroutine which are written in the target file, are commands of the HRJ language.

The other parts of the given portion of translator specification are the same as in the case when MPRJ is the target language. In an analogous way it is possible to alter the complete translator, so that instead of the MPRJ commands, the result of translation is the commands in HRJ language. It is evident from this example that really minimal alterations of the corresponding semantic actions by which translation is performed, are needed, as the structures of MPRJ and HRJ languages are similar.

5. Conclusion

A properties of the translators constructed by the given methodology, enabling their easy adaptation to alterations in the target language, is described. This has been illustrated by using HRJ instead of MPRJ language as the target language. In order the translator could adapt to the changes in the target language, it is necessary to alterate the semantic actions which perform translation of the structure of the source language into the corresponding structure of the target language. In dependence of the fact whether the source and target language have a similar structure or not, smaller or greater alterations of these semantic actions are needed.

A translator can also be adapted to certain alterations in the source

language by appropriate changes in the document for specification, but in the case of a complete change of the source language, the translator has to be constructed from the beginning.

References

[1] Blume C., Jakob W., Pasro. Pascal for Robots, Springer-Verlag, (1985).

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

[3] Meijer H., Nijholt A., YABBER - yet another bibliography: translator writing tools, SIGPLAN Notices, 17, 10, (1982).

[4] 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).

[5] 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).

[6] 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, pp. 107-114, (1992).

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

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

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

[10] 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).

[11] Varga E., Racković M., Construction of a translator for robot-oriented programming language PASRO, XXXVI Yugoslav Conference of ETAN, Kopaonik, (in Serbian), 65-72, (1992).

[12] Varga E., Implementation of the translator for robot-oriented programming languages, Bulletins for Applied Mathematics, BAM 801/92 (LXII), 64-73, (1992).

[13] Varga E., Racković M., Construction of an interpreter for a hypothetic robotic language, 24. Savetovanje proizvodnog mašinstva, (in Serbian), 5-25 - 5-35, Novi Sad, (1992).

REZIME

PRILAGODJAVANJE TRANSLATORA KOSTRUISANOG KOMPAJLER KOMPAJLEROM IZMENI CILJNOG JEZIKA

Konstruisan je translator za robotski orjentisane programske jezike pomoću kompajler kompajlera COCO-2. Ovakav način konstrukcije translatora se pokazao veoma pogodan jer se dobijeni translator veoma lako može prilagoditi izmenama u strukturi polaznog i ciljnog jezika. Prikazane su odgovarajuće izmene translatora u slučaju promene ciljnog jezika.

Received by the editors October 20, 1993