Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 2, February 2014, pg.752 – 760

RESEARCH ARTICLE

Translation of English Algorithm in C Program using Syntax Directed Translation Schema

Nisha N. Shirvi[1], Mahesh H. Panchal[2]

[1]Computer Science & Engineering, Gujrat Technical University, India

[2]Computer Science & Engineering, Gujrat Technical University, India

[1] nisha.shivi@gmail.com; [2] mkhpanchal@gmail.com

Abstract*—*Natural language Processing (NLP) is most promising area of research now days. Automatic Translation Application like Translator of English written algorithm to C program is very useful for the people who want to make programing but don't know any formal language like C, JAVA, etc. Translation of English algorithm to C program has been implemented by Rule based approach using syntax directed translation schema. Rule based system do not use any intermediate representation. The input to the system is naturally written English Algorithms and it output its equivalent C Program. In this paper described system comprises of a two tool Flex (Scanner) and Bison (Parser). Flex as scanner define rules related string acceptance and token generation. Bison as parser define NLP Phrase structural grammar (PSG) with its semantic action.

Keywords*—*Natural Language Processing (NLP); English algorithm; C program; Syntax Directed Translation (SDT) Schema; Translator

I. INTRODUCTION

Writing computer programs in natural language is one of the most obvious yet frustrating tasks in computer science. Such a system sounds incredibly appealing: programming languages are accessible to a small fraction of trained programmers, while natural language is accessible to everyone.

All attempts at natural language programming [2], [18] that is done by restricting the user to a small set of low-level, precise English. This allows systems to produce working code, but the resulting natural language is cumbersome, and everyone still uses formal programming languages with syntax characters. Here describe system, can write programs from natural language which is significantly more abstract, like

Enter two numbers.
Or
Input 2 numbers.
Or
Write two nos.

Although these examples are simpler and smaller, they are also more challenging, because ambiguity is already creeping into the picture. Does 'enter', 'input', 'write' refer to the same meaning? Like 'numbers', 'nos', '2','two'???How to manage these different regular expression generated words and to generate same output for all same meaning words?

The system resolves this ambiguity by making collection of all words together, whose meaning same, means make dictionary for all these words. Generate same token for all these and pass it to parser. In the parser rules related that token written and its related semantic action which makes same output for all these similar meaning words based on syntax directed translation schema.

The two major goals in any translation system development work are accuracy of translation and speed. Accuracy-wise, smart tools for handling translation standards including equivalent words, expressions, and phrases in the target language are to be developed. The grammar should be optimized with a view to obtaining a single correct parse and hence a single translated output. Speed-wise, innovative use of corpus analysis, efficient parsing algorithm and design of efficient Data Structure are required .These thing is very well handle by Bison LALR and GLR parsing algorithm [10] [25] [28] with Faster token supplier Flex [25] [29].

Here describe system cannot eliminate all ambiguity of NLP. System is just an opportunity to potentially learn such formal language without knowing it.

II. LITERATURE REVIEW

This section introduces a concept of natural language supplemented programming languages [18]. And in this paper described system related, small history of pure naturalistic programming system.

A. Natural Language Supplemented Programming Languages

There have ever been programming languages making highly use of natural language setting up their commands. We call this kind of programming natural language supplemented programming. For instance, there are COBOL, FORTRAN and BASIC. Newer examples are KlarDeutsch and AppleScript.

I) KlarDeutsch:

KlarDeutsch, meaning "clear German" is a newer, but unknown and rather simple approach of natural language supplemented programming in the technical domain. KlarDeutsch has been developed by Andover Corporation in 1995.Its purpose is to control machines and equipment with very basic natural language sentences, written in German. A KlarDeutsch program translated to English could look as follows:

```
pumpOff:
pump=off
if heatingValve > 0,4 and faultPump = on then goto pumpOn
if currentDay > 6 then goto block
pumpOn:
pump=on
if heatingValve < 0,2 or faultPump = off then goto pumpOff
block:
pump=off
if currentSecond > 10 or faultPump = off then goto pumpOff
```

KlarDeutsch is extremely simple and somehow old fashioned still using go to-statements. However, it shows that there seems to be a need especially in engineering to use natural language supplemented programming languages in the native languages of the engineers. Engineers cannot always be familiar with the newest developments in software technology. This has caused a gap be- tween modern programming languages used at the universities and quite outdated programming languages in the industry.

KlarDeutsch is currently used at the European Space Observatory Control (ESOC) in Darmstadt to manage the main energy supply facilities, i.e., for crucial applications, showing that natural language supplemented programming is a lot more than an idea far from practical use.

II) AppleScript:

AppleScript is a script language developed by Apple, Inc. in 1993. It is much more elaborated than KlarDeutsch. This is how a simple AppleScript program could look like:

```
tell application "iTunes"
  launch
  say "I am going to play your favorite " & ¬
      "song three times now!"
  set this_track to some track of playlist "Library"
  repeat with counter from 1 to 3
    play this_track
  end repeat
end tell
```

Even though AppleScript makes extensive use of natural language, like with any natural language supplemented programming language, AppleScript programs are rather a kind of linguistic mask reflecting the structures of an ordinary programming language .For instance, if-then-else statements are not expressed via sentences but in the common style of programming languages. In addition to that, there is no way of adding user defined entities, all actions have to be performed on existing objects like sound or application.

On the other hand, AppleScript is one of the few programming languages, which were multilingual, at least for some time. Until version 8.5 of Mac OS, AppleScript programs could be written in several natural languages, among them English, German, Spanish, French, Italian, as well as Japanese and Chinese. Unfortunately, Apple gave up the multilingual approach, because of the complicated costumer support.

B. Pure Naturalistic Programming

Since the sixties there have been attempts to develop programming languages which would allow writing programs in pure natural language.

One of the first and to this day still best approaches towards pure naturalistic programming is NLC described in the paper of Ballard a Biermann from 1979, [3]. They present a domain-specific naturalistic programming language for matrix operations.

Then Metafor is described in [15] and [16], articles of Liu and Liebermann from 2005. They assume that writing a computer program was like telling a story. Thus, they consequently designed an interface, where one can describe a program idea in natural language, using English sentences. Metafor capable of generating Python scaffolding (referred to as the visualization code) from them, though not fully specified programs.

After that in 2006 Pegasus is described in the paper of Knöll, Roman, and Mira Mezini [18]. Pegasus could be seen as advancement of languages like NLC, having a broad linguistic base in order to incorporate semantic descriptions, as also realized in Metafor.

Then macho system is described in [6] and [7] article .As per Author it is the first system to use dual inputs of natural language and examples to write programs, And Substantial improvements in the level of abstraction relative to natural language programming systems, and in complexity and scope relative to example-based programming systems.

It is found that Pegasus is advancement of NLC and Metafor, but its input style look somewhat similar to formal language. It’s just as low level, requires just as much precision from the programmer, and is just as hard to read and understand. And Macho system is different approach to natural language programing. The proposed translation system from English algorithm to C program is more natural into its input style and it is simple rule based approach with syntax directed translation schema.

III. Design and Implementation

This section gives introduction of overall system work flow and brief explanation of how in this paper described system made using Flex, Bison and Gcc compiler, with what is added in Flex and Bison file sections

A. Overview

Most natural language programming systems rely on grammar as a crude replacement for syntax. While a compiler parses a method call and a variable use, a natural language programming system will break down 'Read two numbers' into an imperative verb, determiner, and noun. It will then assign an operation or function to each verb and a variable to each noun. Here the sentence structure is simple and the words are precise, this can be done with a small number of deterministic rules.

Translator of English algorithm into C program has been developed with the following system modules 1. A scanner (Flex) for matching input string and generate related tokens [29]. 2. A parser generator (Bison) for input string syntax checking and generating related semantic action [10]. Then as third module/step a Gcc Compiler link Flex and Bison generated lex.yy.c and Y.tab.c to generate final system executable file, by writing command on CMD like Gcc lex.yy.c Y.tab.c –o TEC.exe. After that as forth module/final step executable file generate output .c file with given input text file, by writing command on CMD like TEC.exe Algo.txt Output.c. (Where Algo.txt is input file and Output.c file is C file related to input file).Work Flow diagram of the same is shown in figure 1.

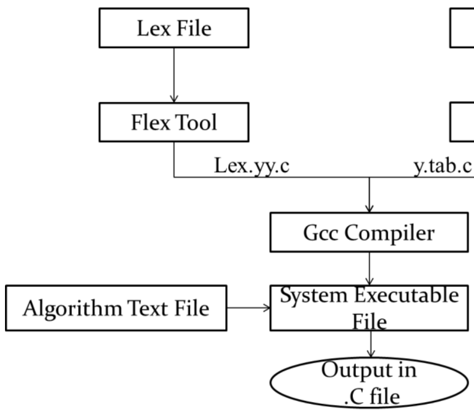

Figure 1: Translator of English algorithm into c program using SDT schema work flow

B. Flex as Scanner or Tokenizer

Scanners may be hand written or may be automatically generated by a lexical analyzer generator (Example Lex, Flex) from descriptions of the patterns to be recognized. The descriptions are in the form of regular expressions, often shortened to regex or, regexp.

A flex program basically consists of a list of regexps with instructions about what to do when the input matches any of them, known as actions. A flex-generated scanner reads through its input, matching the input against all of the regexps and doing the appropriate action on each match. Flex translates all of the regexps into an efficient internal form that lets it match the input against all the patterns simultaneously, so it's just as fast for 100 patterns as for one. Flex produces an entire scanner module that can be compiled and linked to other compiler modules.

An input file for Flex is of the form:
Flex generates a file containing the function yylex () which returns an integer denoting the token recognized.

```
C and scanner declarations
%%
Token definitions and actions
%%
C subroutines
```

The first section of the Flex file contains the C declaration to include the file (TEC.tab.h) produced by Yacc/Bison which contains the definitions of the multi-character tokens. The first section also contains Lex definitions used in the regular expressions.

The second section of the Flex file gives the regular expressions for each token to be recognized and a corresponding action. The third section of the file is empty or may contain C code associated with the actions.

Compiling the Lex file with the command Flex TEC.l (flex filename.lex) results in the production of the file lex.yy.c which defines the C function of yylex (). One each invocation, the function yylex () scans the input file a returns the next token.

In this paper described system add following section into Flex .l extension file:

In first section system must write two include file, one is Bison generated header file and another is string related function header file as system is lie to string world.

In second section added simple small dictionary for system with its related action. Here in this application is its first phase so with small dictionary implementation done, without making generic regexps. Like
"Print" | "is"   { yylval.string_val = strdup (yytext);return V; }

Where 'print, is ' regexp for system, and 'yylval.string_val = strdup(yytext);return V;' action part means its convert by default integer recognized tokens into system required string tokens, where yylval and yytext Flex predefined variable for pointer to matched string from input file and value associated with token.

In third section yywrap () function written, for more information see Flex functions table.

Table1: Flex variables

Yyin	Of the type FILE*. This point to the current file being parsed by the lexer.
Yyout	Of the type FILE*. This point to the location where the output of the lexer will be written. By default, both yyin and yyout point to standard input and output.
Yytext	The text of the matched pattern is stored in this variable (char*).
Yyleng	Gives the length of the matched pattern.
Yylineno	Provides current line number information. (May or may not be supported by the lexer.)

Table2: Flex functions

yylex()	The function that starts the analysis. It is automatically generated by Flex.
yywrap()	This function is called when end of file (or input) is encountered. If this function returns 1, the parsing stops. So, this can be used to parse multiple files. Code can be written in the third section, which will allow multiple files to be parsed. The strategy is to make yyin file pointer point to a different file until all the files are parsed. At the end, yywrap () can return 1 to indicate end of parsing.
yyless(int n)	This function can be used to push back all but first 'n' characters of the read token.
yymore()	This function tells the lexer to append the next token to the current token.

C. Bison as Parser Generator

A parser is a program which determines if its input is syntactically valid and determines its structure. Parsers may be hand written or may be automatically generated by a parser generator from descriptions of valid syntactical structures. The descriptions are in the form of a context-free grammar.

Bison is a general-purpose parser generator that converts an annotated context-free grammar into a deterministic LR or generalized LR (GLR) parser employing LALR (1) parser tables. As an experimental feature, Bison can also generate IELR (1) or canonical LR (1) parser tables. Once you are proficient with Bison, you can use it to develop a wide range of language parsers, from those used in this paper describe system made.

Bison is upward compatible with Yacc: all properly-written Yacc grammars ought to work with Bison with no change. Anyone familiar with Yacc should be able to use Bison with little trouble Java is also supported as an experimental feature.

An input file for Bison is of the form:

```
C and parser declarations
%%
Grammar rules and actions
%%
C subroutines
```

The first section of the Bison file consists of optional ordinary C subroutine declaration part, a list of tokens (other than single characters) that are expected by the parser and the specification of the start symbol of the grammar also done. This section of the Bison file may contain specification of the token recognizer return type as shown below. This permits greater flexibility in the choice of a context-free grammar and its token return value.

The second section of the Bison file consists of the context-free grammar for the language. Productions are separated by semicolons, the '::=' symbol of the BNF is replaced with ':', the empty production is left empty, non-terminals are written in all lower case, and the multi character terminal symbols in all upper case. Within the braces for the action associated with a production is just ordinary C code. If no action is present, the parser will take no action upon reducing that production.

The third section of the Yacc file consists of C code. There must be a main () routine which calls the function yyparse (). The function yyparse () is the driver routine for the parser. There must also be the function yyerror () which is used to report on errors during the parse. And other ordinary C function definition.

Compiling the Bison file with the command bison -dv TEC.y (bison -dv filename.y) causes the generation of two files TEC.tab.h and TEC.tab.c. The TEC.tab.h contain list of tokens is included in the file which defines the scanner. The TEC.tab.c defines the C function yyparse () which is the parser.

In this paper described system add following section into Bison .y extension file:

In first section system write general C code required include file , variable declaration and user define function protocol in between "%{" " %} " braces. After that, below these braces and before first %% division, %union (which defines the structure that the lexer will use to pass lexemes to the parser) bison specification write to change string denoting token Tokenizer instead of by default flex integer Tokenizer. And few bison specification listed in table Bison specification to make complete system.

In second section add phrase structural grammar for English language with its simple C coded action to generate output based on syntax directed translation schema concept, as production of grammar match, output is generated related production. Like

```
Sentence: np vp {printf ("Noun Phrase is %s and verb phrase is %s", $1, $2)}
        ;
```

Where 'Sentence: np vp 'is phrase structural grammar one production, ended with ';' and {printf ("Noun Phrase is %s and verb phrase is %s", $1, $2)} is action related sentence production.

In third section main () function written with yyerror () and user define function which used in grammar action part and main function for making system complete.

Table 3: Bison specification

Bison Specification	Specification Description
%union	Defines the Stack type for the Parser. It is a union of various datas/structures/objects.
%token	These are the terminals returned by the yylex function to the yacc. A token can also have typ e associated with it for good type checking and syntax directed translation. A type of a token can be specified as %token<stack member>token Name.
%type	The type of a non-terminal symbol in the Grammar rule can be specified with this. The forma t is %type <stack member> non-terminal.

%noassoc	Specifies that there is no associativity of a terminal symbol.
%left	Specifies the left associativity of a Terminal Symbol
%right	Specifies the right associativity of a Terminal Symbol.
%start	Specifies the L.H.S non-terminal symbol of a production rule which should be take-n as the starting point of the grammar rules.
%prec	Changes the precedence level associated with a particular rule to that of the followi-ng token name or literal.
%glr-parser	To make parser algorithm is GLR
%error-verbose	to get more specific error

IV. Conclusions Result and Discussion

Evaluate the System there no standard benchmark suite that would allow system to compare its results against other systems. Therefore System is tested with simple test suite. In this the system was tested with basic C concept like scanf(),printf(),if(),if() else(),for(),automatic variable , array and pointer declaration. The system returned correct meaningful translations in all of the cases. A group of sample input sentences with the tabulated outputs are shown in table 4 to give a correct picture of the results obtained.

The system takes care of similar meaning words ambiguity based on lexical category successfully. The word case sensitivity is not handled by the system as the small dictionary is considered and no morphological phase added. The system output can be enhanced including larger dictionary and morphological phase.

Table 4: Group of tabulated results

System Input	System Output

read two numbers.
add two numbers.
display answer.
or
input two nos.
add two nos.
show answer.
or
write two numbers.
addition of two numbers.
Output answer.
or
all basic math function program like subtraction, multiplication, division etc.
	#include <stdio.h>
	#include <string.h>
	void main()
	{
	int numbers1 ;
	int numbers2 ;
	int answer=0;
	scanf("%d",&numbers1);
	scanf("%d",&numbers2);
	answer = numbers1 + numbers2 ;
	printf("answer is %d",answer);
	getch();
	}

enter your name.
enter your qualification.
enter your designation.
enter your age.
print name.
print qualification.
print designation.
print age.
	#include <stdio.h>
	#include <string.h>
	void main()
	{
	char *name;
	char *qualification;
	char *designation;
	int age;
	gets(name);
	gets(qualification);
	gets(designation);
	scanf("%d",&age);
	printf("name is %s",name);
	printf("qualification is %s",qualification);
	printf("designation is %s",designation);
	printf("age is %d",age);
	getch();
	}

read four numbers.	#include <stdio.h>
add four numbers.	#include <string.h>
display answer.	void main()
	{
	int numbers[4],i=1;
	int answer=0;
	for(i=1;i<=4;i++)
	{
	scanf("%d"\n,&numbers[i]);
	}
	for(i=1;i<=4;i++)
	{
	answer = answer + numbers[i] ;
	}
	printf("answer is %d",answer);
	getch();
	}
read two numbers.	#include <stdio.h>
if number1 is greater than number2.	#include <string.h>
print number1 is greater.	void main()
else number2 is greater.	{
	int numbers1 ;
	int numbers2 ;
	scanf("%d",&number1);
	scanf("%d",&number2);
	if(number1 > number2)
	{
	printf("number1 is greater);
	}
	else
	{
	printf("number2 is greater");
	}
	getch();
	}

V. Conclusion and Future Work

In this paper , discussed a Translation of English algorithm into C program using syntax directed translation schema system, and how combining Flex and Bison tool makes it easier to generate correct solutions while also reducing the ambiguity in abstract natural language. System can correctly generate all C basic concept like scanf(),printf(),if() ,if()else(),for(),automatic variable , array and pointer declaration.

Adding larger dictionary of larger lexicon and morphological phase into system more concept of C program correctly generated. And in this paper described system can easily be modified to build translation systems for other language, making Tokenizer for other natural language like HINDI, CHINES, etc. and system grammar production action part for other formal language like JAVA, C# etc.

References

[1] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman "*Compilers: Principles, Techniques, and Tools*" Publisher: Addison Wesley. (1986): 796

[2] Biermann, Alan W., and Bruce W. Ballard. "*Toward natural language computation.*" Computational Linguistics 6, no. 2 (1980): 71-86.

[3] Ballard, Bruce W., and Alan W. Biermann. "*Programming in natural language: "NLC" as a prototype*." In Proceedings of the 1979 annual conference, pp. 228-237. ACM, 1979.

[4] Carlos, Cohan Sujay. "*Natural Language Programming Using Class Sequential Rules*." In IJCNLP, pp. 237-245. 2011.

[5] Christiansen, Morten H., and Nick Chater. "*Connectionist natural language processing: The state of the art.*" Cognitive science 23, no. 4 (1999): 417-437.

[6] Cozzie, Anthony, and Samuel T. King. "*Macho: Writing programs with natural language and examples.*" Technical report, University of Illinois at Urbana-Champaign, 2012.

[7] Cozzie, Anthony, Murph Finnicum, and Samuel T. King. "*Macho: Programming with man pages*." Proceedings of the 2011 Workshop on Hot Topics in Operating Systems (HotOS 2011). 2011.

[8] D.Jurafsky, J.H Martin. *Speech and natural language processing*. India: Pearson Education, 2000, pp 657-671.

[9] Dan W. Patterson, *Introduction to Artificial Intelligence and Expert System,* PHI, 2001, Chapter 12.

[10] Donnely, C., and Richard Stallman. "*Bison: The Yacc-compatible parser generator, 2006.*" (2008).

[11] Elaine and Kevin Knight, *Artificial Intelligence*, McGraw Hill companies Inc., 2006, Chapter 15.

[12] Igo, Sean, and Ellen Riloff. *"Learning to Identify Reduced Passive Verb Phrases with a Shallow Parser*." In AAAI, pp. 1458-1461. 2008.

[13] Enju syntactic parser Online demo http://www.nactem.ac.uk/tsujii/enju/

[14] Han, Aaron Li-Feng, et al. "*Phrase Tagset Mapping for French and English Treebank's and Its Application in Machine Translation Evaluation*." Language Processing and Knowledge in the Web. Springer Berlin Heidelberg, 2013. 119-131.

[15] Liu, Hugo, and Henry Lieberman. *"Metafor: Visualizing stories as code."* In Proceedings of the 10th international conference on Intelligent user interfaces, pp. 305-307. ACM, 2005.

[16] Liu, Hugo, and Henry Lieberman. "*Programmatic semantics for natural language interfaces*." In CHI'05 Extended Abstracts on Human Factors in Computing Systems, pp. 1597-1600. ACM, 2005.

[17] K.R.Chowdhary,*Natural language processing,* http:/krchowdhary.com/me-nlp/nlp-01.pdf

[18] Knöll, Roman, and Mira Mezini. "*Pegasus: first steps toward a naturalistic programming language*." In Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, pp. 542-559. ACM, 2006.

[19] Tomita, Masaru. *" Efficient parsing for natural language: a fast algorithm for practical systems*." Vol. 8. Springer, 1985.

[20] McPeak, Scott, and George C. Necula. "*Elkhound: A fast, practical GLR parser generator*." In Compiler Construction, pp. 73-88. Springer Berlin Heidelberg, 2004.

[21] Mihalcea, Rada, Hugo Liu, and Henry Lieberman. *"Nlp (natural language processing) for nlp (natural language programming)*." In Computational Linguistics and Intelligent Text Processing, pp. 319-330. Springer Berlin Heidelberg, 2006.

[22] Nair, Latha R., David Peter, and Renjith P. Ravindran. "*Design and Development of a Malayalam to English Translator–A Transfer Based Approach.*" Int. J. of Computational Ling. (IJCL) 3.no 1 (2012).

[23] *NLPA-Syntax* , www.cs.bham.ac.uk/~pxc/nlp/NLPA-Syntax.pdf

[24] RC Chakraborty, *Natural Language Processing,* www.myreaders.info.

[25] Simões, Alberto, Nuno Carvalho, and José João Almeida. "*Generating flex lexical scanners for perl parse:: Yapp*." (2012).

[26] Stanford Parser Online demo , http://nlp.stanford.edu:8080/parser/

[27] Stanford Corenlp *Tool* Online demo http://nlp.stanford.edu:8080/corenlp/

[28] Stephen C. Johnson. *Yacc: Yet Another Compiler-Compiler.* Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

[29] Paxson, V. *"GNU Flex Manual, Version 2.5. 3*." (1996).

[30] Wikipedia.*Backus–naur form.* Web site: http://en.wikipedia.org/wiki/Backus- Naur_Form [Last accessed: 19-03-2012].