# ON THE MACHINE TRANSLATION OF PROGRAMMING (ARTIFICIAL) LANGUAGES INTO NATURAL LANGUAGES

*Santa Bdr. Basnet, Shailesh Bdr. Pandey and Yogesh Raj*

**0.** We begin this paper by declaring our conviction that the translation of any formal/programming (artificial) language into a natural language and vice-versa is not only possible, but I actually desirable. This conviction is based on our notion of *appropriate* Information Technology systems where the systems do not intrinsically offer to a user the barriers to its accessibility i.e. the systems do not discriminate users on the *sole* basis of their knowledge of their formal/programming languages. These systems are adaptive in a sense that they shall have an in-built component called a *flexible translator*, which will translate the user's natural language inputs to a programming language and vice-versa. This paper is a tiny step towards the formulation of such a flexible translator.

**0.0.** The ultimate aim of this line of research is, however, to lay down the principles of intelligent machines that work in a natural language environment. We wish to claim here in passing that we consider machine intelligence is an emergent property that necessarily *follows* natural language comprehension and production.

**1.** The *translation* is not the *enumeration* of the action that the syntax of the Artificial language implies. Although a code in a programming language is *meant for* an implied action. For example, the *translation* of (1a) such that

(1a) b=c%d;p=a*b;

is not

(1b) *divide c by d; retain the remainder; assign the remainder to b, multiply by a; p equals the product.*

Although this is what a C-compiler performs upon being supplied with the expression.

**1.1.** The difference between a natural language and an artificial language seems to be the *formality* of the latter. The *formality* excludes anything other than the computational sequences which any expression *triggers*. This exclusive function puts a severe limit to the translatability of a formal language, as here, its translatability is measured by *translating* the syntax into *actions.*

**1.2.** Translatability has the fuzzy (shifting/overlapping) semantic field-boundaries as its pre-requisite, and devoid of the fuzziness (as in all cases of context-free grammars), *translation* has a restricted meaning of *enumeration*. For illustrating this point, considering a mathematical expression (2a):

(2a) $x= [2+8 \div (10-6) \times 4]$

The *translation* of this expression is neither (2b)

(2b) $x=10$.

Nor (2c),

(2c) $x1 = (10-6)$;
$x2 = 8 \div x1$;

```
x3 = x2 × 4;
x4 = 2 +x3;
x = x4.
```

and nor (2d).

(2d) *Subtract 6 from 10. Divide 8 by the remainder. Multiply the quotient of 4. Add the product to 2. Assign the sum as x.*

Here (2c) is a mathematical and (2d) is a non-mathematical *enumeration*.

**2.** Another distinguishable feature of the formal language is that all of their expressions are ‘imperative sentences’. Seemingly declarative statements too are imperative sentences with a command to declare (something). For example, consider (3a).

(3a)
```
{
    int a;
}
```

(3a) is a declarative statement which declares *a* to b an integer. The statement, however, is actually a command to the computer to *declare a* as an integer.

Similarly,

(3b)
```
{
    p=n;
}
```

(3b) is actually a command to *equalize n* to *p*.

**2.1.** Thus all *operator* characters and *keywords* of a formal language are ‘main verbs’ of its formal ‘linguistic expressions’.

**2.1.1.** No formal linguistic expression has ‘auxiliaries’, or pre- or post-verbal elements, hence the qualifier ‘main’ in the expression ‘main verbs’ is practically redundant.

**2.2.** Important is also the point that although a formal ‘linguistic expression’ may have more than one operator character and/or keywords, an overtly or covertly marked *logical ordering* ensures that no two ‘verbs’ are simultaneous. Verb serialization is an *inherent feature* here (as compared to aspectual markers in natural languages) with ‘verbs’ to recede are marked differently that the ‘verbs’ to proceed.

**2.2.1.** Even for a command for a simultaneously (as in the case of parallel processing), such is achieved by creating parallel ‘worlds’, *albeit* temporarily.

**3.** The basic work we are going to report here is done by us at Nepal Engineering College, Bhaktapur. The project has the following routine:

(1) ‘Read’ and parse of a formal (here C-language) ‘linguistic expression’ into ‘words’;
(2) Translate the ‘words’ into ‘word of *machine English*’;
(3) Produce comparable *machine* English expressions;
(4) Operate *grammatical* rules on the expressions of the *machine* English to output those of the ‘*natural English*’, and
(5) Vice-versa.

**3.1.** The term *machine English* is used here to denote a kind of 'rudimentary' English that is constituted of the 'empty slots' filled up by English-like 'words' while its word order remain unchanged as compared to the C. See (4a) and (4b).

| | | |
|---|---|---|
| (4a) | C-language: | int a,b,c,d; |
| | *machine* English: | integer a and b and c and d. |
| (4b) | C-language: | var1=var2; |
| | *machine* English: | variable var1 assign variable var2. |

**3.2.** The term *natural English* is used here to denote a kind of English that has acceptable differences to the *common spoken English* (typically called so by COBUILD or OED editorial groups) with *grammatically correct* sentence structures.

**3.3.** The difference between *machine* and *natural* variety of English may remind some of us similar differences known to exist between *deep structure* and *surface structure* of a natural language. The similarity can further be extended by saying that natural English is generated from the machine English by applying a number of *grammatical rules* (or, *transformational rules*). Such as (5a) and (5b).

| | | |
|---|---|---|
| (5a) | *machine* English: | integer a and b and c and d. |
| | *transformational rule*: | (1) Topicalization (2) Foregrounding/Phrase Embossing. |
| | *Natural* English: | define a,b,c and d as integers. |
| (5b) | *machine* English: | variable var1 assign variable var2 |
| | *transformational rule*: | (1) Topicalization (2) Redundant Conversion. |
| | *Natural* English: | assign variable var2 to var1. |

**4.** LITERATUR REVIEW. A project by Leevi Marttila [2000] ha the title "C to English to C translator c2txt2c v0.1" [ibid] which was later changed to "Accurate language to inaccurate language (and back) translator". This project translated the C source code into English that had no apparent relation to the original code. The hexadecimal number for example is converted into poems in this project. The project was mainly inspired by the treatment of source code with respect to the first amendment law of the US. The project wanted to demonstrate that C source was a form of speech and not just a "functional device like a telephone circuit" as stated by Judge Gwin of the Federal District Court of the Northern State of Ohio. The translation was thus just a form of encryption rather than natural language understanding.

Later, Schwartz [2001] wrote c2eng, which translated the C into a text that described the actual functioning of the program. The program however, did not translate generated functional statements back into C code. Also the English language generated was not natural English as the output was heavily loaded with jargons not found in a Standard English dictionary.

Baccas [2001] developed a program named BabelBuster. It was also inspired by the treatment of source code with respect to the First Amendment law of U.S.A. BabelBuster

was created with a vision that C source code can be machine translated into grammatically correct English that is easy to read, and the output can again be machine translated back into C. This project also translated C code into English that was not free of words of common usage. It included C specific words such as “printf”, ”%d” and “\n”.

**5.** PROJECT DETAILS

**5.1.** SOFTWARE SPECIFICATION AND REQUIREMENTS

| | | |
|---|---|---|
| Platform | : | Win-32 Platform with MFC support |
| Programming Language | : | Microsoft Visual C++ 6.0 |
| Developer | : | Microsoft Corporation |
| System Capabilities | : | Enough memory and processor capabilities to support and run Win32 application |

**6.** PROJECT BREAKDOWN

C PARSER. The lexical analysis phase reads character by character of the source program and groups them into a stream of tokens in which each token is an identifier, keyword and punctuation character. The character sequence forming a token is called a lexeme for the token. The syntax of C language like other languages is described by context free grammars or BNF notation. The project uses the BNF notation of the C language.

C TO DEEP STRUCTURE ENGLISH. BNF grammar finds out what type of statement it is. The type of statement dictates the structure of the English sentence the program generates. In our case, they are exclusively the imperatives. The verb is assigned for each type of C statement. For example *define* is used for variable declarations, *assign* for assign statements. Also, the other identifiers in the C statement are replaced with an equivalent word or words. For example comma (,) is replaced by *and* in case of variable declaration. This forms the deep structure for the English sentence to be generated.

MACHINE ENGLISH TO NATURAL ENGLISH. After a deep structure is generated, we apply transformational rules to the deep structure to order the position of the tokens. These transformations are derived so as to generate grammatically correct sentences.

ENGLISH TO C. Transformational rules are applied in reverse order to the generated English sentences to get back C statements.

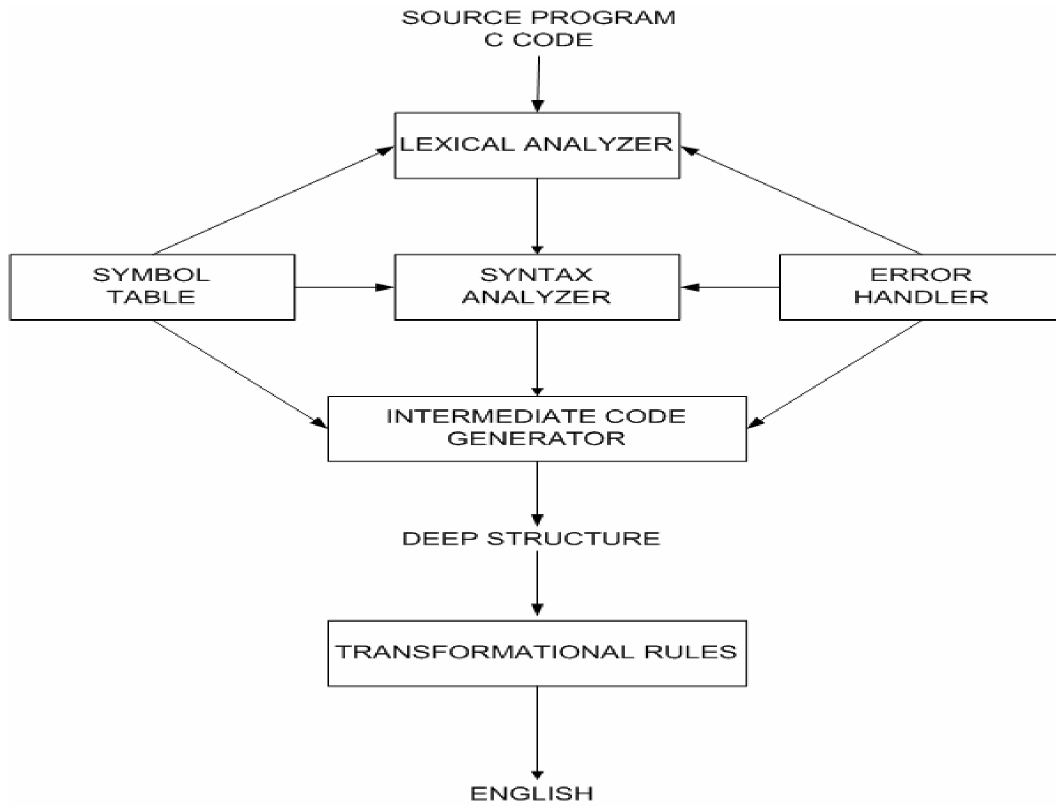

**FIG 1: STEPS IN C TO ENGLISH TRANSLATION**

## **7.** DATABASE FOR KEYWORDS AND OPERATORS

### **7.1.** THE TABLE SHOWS IDENTIFIERS AND KEYWORDS OF C ALONG WITH THEIR SUBSTITUTION

| far | look-far |
|---|---|
| float | real |
| for | for |
| goto | go-to |
| huge | look-farther |
| if | if |
| ifndef | if-not-defined |
| include | include |
| Int | integer |
| line | count-line |
| long | large-integer |
| near | look-close |
| register | register |
| return | return |
| short | small |
| signed | keep-sign- |
| sizeof | measure |
| static | preserve |
| struct | group |

| | |
|---|---|
| switch | select |
| typedef | defined-type |
| undef | de-define |
| union | unite |
| unsigned | modulus-of- |
| void | keep-type-of |
| volatile | temporarize |
| while | while |

**7.2.** THE TABLE SHOWS OPERATORS IN C ALONG WITH THEIR SUBSTITUTION

| ID | NAME |
|---|---|
| + | add |
| - | subtract |
| * | multiply |
| / | divide |
| % | remainderize |
| ~ | complimentize |
| = | assign |
| == | equate |
| != | not-equate |
| ! | not- |
| & | bitwise-and |
| \| | bitwise-or |
| > | greater-than- |
| < | less-than- |
| >= | greater-or-equal |
| <= | less-or-equal |
| ? | if |
| : | then |
| ; | . |
| , | and |

**8.** CONCLUSIONS. The project was able to develop a parse for C statements using a BNF grammar. The project demonstrated the generation of deep structure for some of the C statements by replacing technical jargons by word or words accepted by common English. The project demonstrated the application of transformational rule or rules to generate imperative sentence as the output. By application of reverse transformation the project demonstrated back translation of previously generated English sentences into C.

**9.** IMPLICATIONS FOR FURTHER RESEARCH. 1. The parser currently understands only limited C statements. The project can be extended to parse every C construct. 2.

Semantic analysis can be used to extract features from a sentence to search the database for a match. 3. Program and user interaction system can be provided as a forum for teaching new sentences and clearing doubts in case of uncertainty.

## REFERENCES


Aho, Alfred V.; Ravi Sethi and Jeffrey D. Ullman. 2001. *Compilers: Principles, Techniques and Tools*.: Delhi: Pearson Education.

Allen, James. 2003. *Natural Language Understanding*. Singapore: Pearson Education P. Ltd.

Baccash, Jonathan, BabelBuster – C to English to C Translator. http://www-2.cs.cmu.edu/~dst/DECSS/Baccash/. August 17, 2004.

Catherine, N. Ball. Introduction to computational linguistics. http://www.georgetown.edu/faculty/ballc/ling361/ling361_aboutcl.html. September 8, 2004.

*COLLINS COBUILD: Dictionary of Phrasal Verbs.* 1995. William Collins Sons & Co. Ltd.

Kragen, Sitaker. Recursive-descent parsers for BNF grammars. http://lists.canonical.org/pipermail/kragen-hacks/1999-October/000202.html. August 21, 2004.

Hopcroft, John E. Rajeev Motwani; and Jeffrey D. Ullman. 2001. *Introduction to automata theory, languages and computation.* Singapore Addison Wesley Longman P. Ltd.

Marttila, Leevi. Accurate language to inaccurate language (and back) translator c2txt2c v0.2.1. http://personal.sip.fi/~lm/c2txt2c/. August 17, 2004.

Smith, Ronnie W.; D. Richard Hipp. 1994. *Spoken Natural Language Dialog Systems.* 200 Madison Avenue. New York: Oxford University Press Inc.

Tizzard, Keith. 1986. *C for Professional Programmers*. New Delhi: East-West press.

Schwarz, Omri. C to English to C in Perl, http://www.mit.edu/~ocschwar/C_English.html. August 17, 2004.

Uszkoreit, Hans. http://www.coli.uni-sb.de/~hansu/. What is Computational Linguistics? September 8, 2004.

http://www.wordiQ.com/. Definition of computational linguistics. September 8, 2004.

http://www.tranexp.com/, October 2, 2004.

http:// www.translation-experts.com/, October 2, 2004.