An Intelligent NC Program Processor for CNC System of Machine Tool

Yadong Liu[1], Xingui Guo[1], Wei Li[1], Kazuo Yamazaki[1], Keizo Kashihara[2], Makoto Fujishima[2]

[1]IMS-Mechatronics Laboratory, Mechanical Aeronautical Engineering Department, University of California, Davis, USA

[2]Mori Seiki Co. Ltd., 2-35-16 Meieki, Nakamura-ku, Nagoya City, Japan

Abstract

NC program interpreting is one of the most important tasks of CNC in machine tool system. The existing CNC systems only support vendor-specific NC program input, which restrict the applying of other similar functional NC programs with different program format. Especially for those users owning several machine tools with different CNC from the same provider, the diversity of NC programs dramatically increases their cost and time on operator training and machine tool maintenance. In order to deal with the variety of NC program, an intelligent NC program processor (NCPP) is proposed in this paper. The proposed NCPP has a novel structure with separated NC specification dictionary and processing engine, which ease the CNC system to adapt to a new NC program specification by only updating the NC specification dictionary (NCSD). This paper explains the NC specification dictionary structure by using EBNF representation, which is further implemented in TCL, an embedded script language tool. Meanwhile, the unchanged processing engine has been designed from a compiler's point of view to process the data dictionary. A NCPP prototype is built to evaluate the proposed design. Case study shows that the proposed NCPP handles different NC program inputs successfully.

Keywords: NC Program Processor, Canonical Machining Function, EBNF, TCL

1 INTRODUCTION

In the CNC system of modern machine tool, NC program interpreting is very important, which is in charge of the accurate resolving of machining intention generated from CAM system. Figure 1 shows the role of NC program processor (NCPP) in CNC system, the input is NC program. The major function of NCPP is to decode the input into motion command and PLC command, and send them to the motion control processor (MCP) and programmable logic controller (PLC) of CNC separately in order to control the movement of the cutting tool and auxiliary machine logic. Most CNC systems can handle only one specific NC program format, while the diversity of NC programs always entangles the machine tool users, especially for those owning several machine tools with different CNC but from the same provider. Such a situation is formed due to the following facts:

First, no actual NC program standard is available, although some nominal ones are existing. There have been three basic standards: RS274D (USA), ISO6983 (ISO) and DIN66025 (Europe) for NC program since CNC was invented. However, machine tool and control technology have undergone great development since then, thus a lot of new functions and controller-specific features, not supported by these NC standards have been added to the CNC.

Second, different CNC providers extended the NC program standard a lot to adapt their own specific functions; a typical example is the special interpolation function added by each CNC provider. This contributes to the difficulty of interchanging NC programs generated by different CNC systems. Hence, even the users who own several kinds of machine tools from the same machine tool manufacturer but equipped with different CNC systems have to generate different NC programs for each machine tool. This situation even happens in the same CNC system case with later updated functions.

Third, NC program modification is always mandatory in order to safely reuse a NC program with different format. This process usually requires much more specialist knowledge of different kinds of CNC, although the CNC providers always say that only less than 5% change is needed. This issue obviously forces the machine tool users to spend more expense and time on training their programmers or operators.

Fourth, most CAM system can generate different NC program by applying different post-processor now. However, it’s not workable for manual programming cases, which still cover major applications in industry. In addition, different CNC system requires different post-processor, the CAM system thus has to prepare a lot of unused post-processors in advance. Universal post-processor is another solution, which allows the end user to customize a post-processor; however making a specific post-processor requires much more experienced specialists.

In order to efficiently reuse NC programs from different CNC systems, an intelligent NCPP is proposed to deal with variety of NC program inputs in CNC system. With separated processing engine and NC specification dictionary (NCSD) structure, the proposed NCPP can easily interpret different NC programs by applying different specification dictionaries while keeping the processing engine unchanged.

The rest portions of this paper are organized as following. In Section 2, the interface of NCPP is introduced. In Section 3, conceptual model of proposed NCPP is presented. Section 4 discusses the design of proposed NCPP. Section 5 shows the implementation of a prototype system. The paper is concluded in Section 6.

2 INTERFACE OF NC PROGRAM PROCESSOR

NCPP is one module of the CNC, which requires cooperation between different modules; therefore it’s quite necessary to clarify the interface before starting design. As shown in Figure 1, the purpose of NCPP is to translate the input NC program into machine instruction, such as motion command, PLC command or simple parameter settings and error messages. NIST calls these outputs as Canonical Machining Functions. A set of definition of these canonical machining functions has been given as shown in table 1, which has been used by Enhanced Machine Controller (EMC) project and other open CNC research projects.

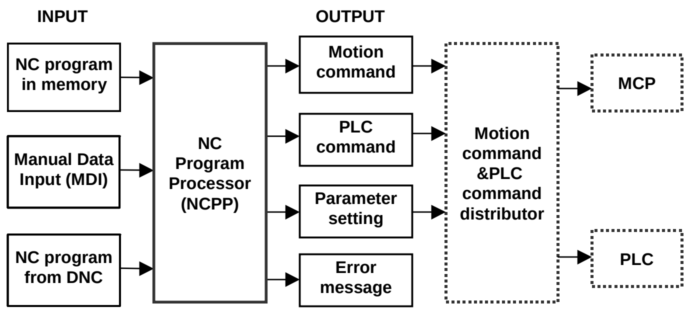


Figure 1: Role of NCPP in CNC system.

The canonical machining functions were devised with two objectives in mind:

- All the functionality of common 3-axis to 5-axis machining centers had to be covered by the functions; for any function a machining center can perform, there has to be a way to tell it to do that function.
- It must be possible to interpret RS274-compatible NC program into canonical machining function calls.

Table 1: Canonical machining function definitions.

Functionality	**Functions**

Representation	SET_ORIGIN_OFFSETS(x,y,z,a,b,c)
	USE_LENGTH_UNITS(units)

Free Space Motion	STRAIGHT_TRAVERSE(x,y,z,a,b,c)

Machining Attributes	SELECT_PLANE(plane)
	SET_FEED_RATE(rate)
	SET_MOTION_CONTROL_MODE(mode)
	……

Machining Functions	ARC_FEED(first_end, second_end, …)
	DWELL (seconds)
	STRAIGHT_FEED(x,y,z,a,b,c)

Spindle Functions	SET_SPINDLE_SPEED(r)
	START_SPINDLE_CLOCKWISE()
	STOP_SPINDLE_TURNING()

Tool Functions	CHANGE_TOOL(slot)
	SELECT_TOOL(i)

……	……

The canonical machining functions are atomic commands. Each function produces a single tool motion or a single logical action. A NC command usually includes two types: those for which a single NC command corresponds exactly to a canonical function call and those for which a single command will be decomposed into several canonical function calls. Things like "move in a straight line” or “turn flood coolant on” are of the first type. Things like “run a peck drilling cycle” are of the second type.

Beside NC program input and canonical machining functions output, there are other four kinds of interface functions are required as NCPP is running to interpret input NC program, as shown in Figure 2:

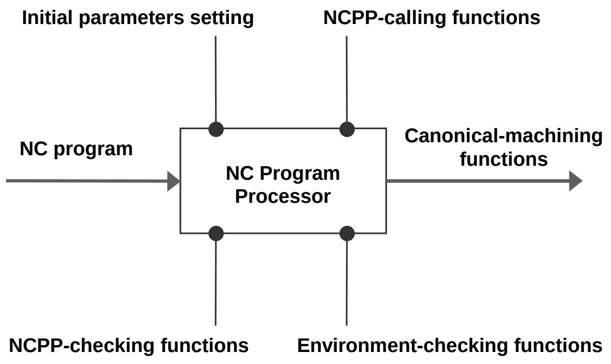

Initial parameters setting
NCPP-calling functions
NC program
NC Program Processor
Canonical-machining functions
NCPP-checking functions
Environment-checking functions

Figure 2: Interface of NCPP.

- Initial parameters setting, to establish an NCPP environment, such as loading default parameter files and tool tables.
- NCPP-calling functions, to tell the NCPP to do something, such as reading a NC block or executing the block last read.
- NCPP-checking functions, to ask the NCPP to give information, such as inquiring current active G codes, M codes or error code.
- Environment-checking functions, to ask the world outside the NCPP to give the NCPP information, such as current system feedrate, spindle speed, tool table and so on.

Among those interface functions, the corresponding relation between inputting NC program and outputting canonical machining functions is the key point of this paper.

3 CONCEPTUL MODEL OF PROPOSED NCPP

Figure 3 show the proposed NCPP concept. Compared to the traditional design, the major feature of this NCPP is the structure with separation of NCSD and processing engine. Within this NCPP, different NC program could be interpreted in terms of different NCSD, while the processing engine keeps the same. For example, suppose the input NC program follows Fanuc specification, the engine will refer to the Fanuc NCSD to do interpretation. Next time, if a NC program following Mitsubishi specification is given, the same engine will refer to the Mitsubishi NCSD to interpret it. For the two cases, it can be seen that each time only different NCSD is chosen, while the processing engine does not change.

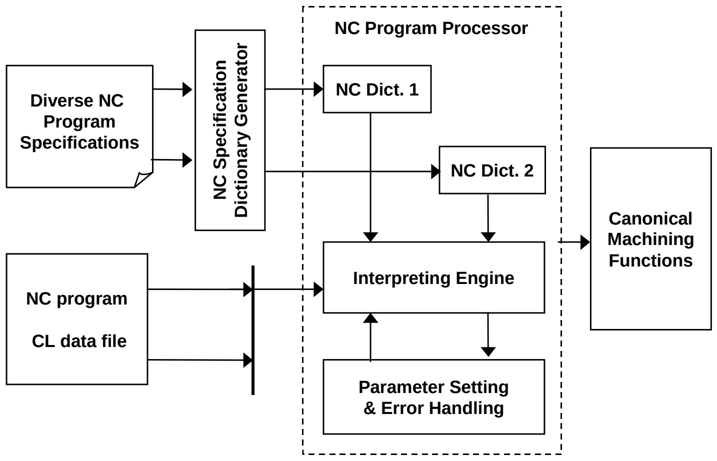

NC Program Processor
Diverse NC Program Specifications
NC Specification Dictionary Generator
NC Dict. 1
NC Dict. 2
Canonical Machining Functions
NC program
CL data file
Interpreting Engine
Parameter Setting & Error Handling

Figure 3: Conceptual model of proposed NCPP.

Such a solution provides dramatic flexibility and stability for the NCPP development, only one set of software code of the processing engine needs to be maintained. Even if there is an input NC program following a NC specification which is not available in the existing NCSDs, a new NCSD can be generated and added easily without recompiling the source code of the processing engine.

To implement the proposed NCPP, two key issues should be solved: how to define these dictionaries and how to design the engine. That is the focus of this research, which will be explained in the following section.

4 DESIGN OF PROPOSED NCPP

Based on the conceptual model of the proposed NCPP shown in Figure 3, the NCPP structure is given in Figure 4.

If looking inside the NCPP, the key portion is the interpreting (processing) engine mentioned before, from a compiler's point of view, the engine can be divided into four steps in order to check and decode an input NC program.

These four steps are:

- Lexical analysis, which checks the character-based error within a NC program;

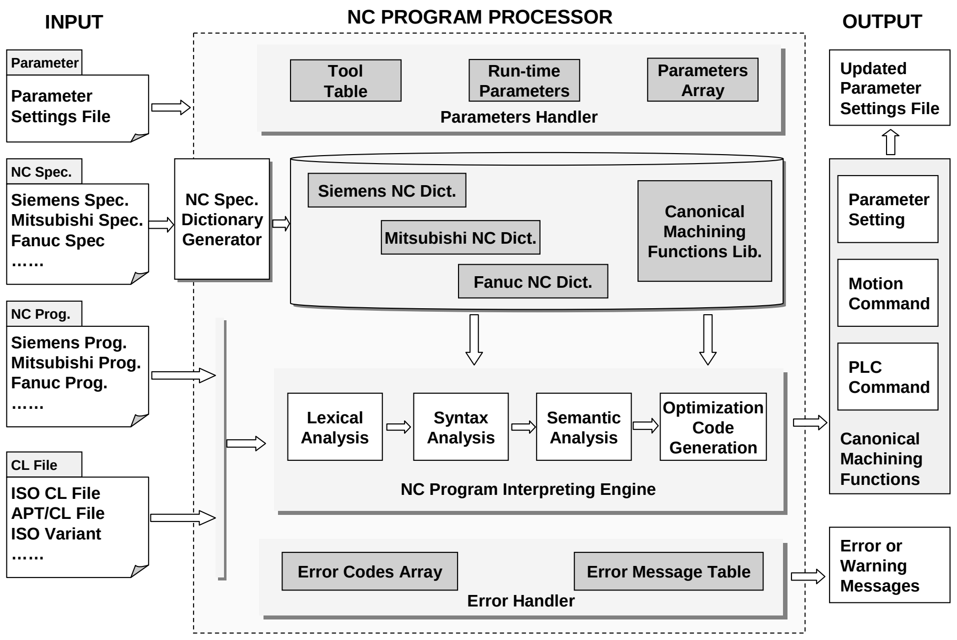
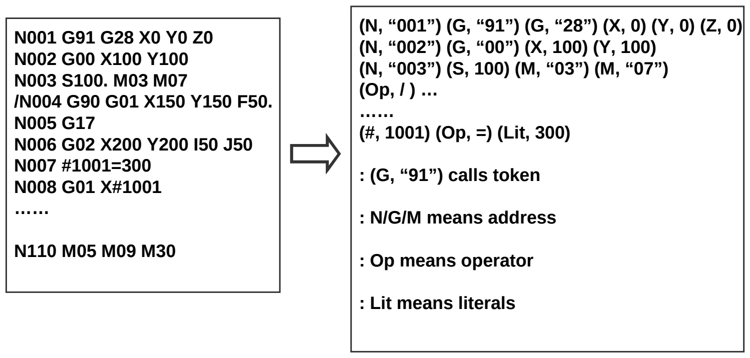


Figure 4: Proposed NCPP structure.

- Syntax analysis, which makes sure the logic relation within each block of NC program is correct;
- Semantic analysis, which checks the inter-block logic correctness of a NC program.
- Optimization & code generation, which decode block and generate the canonical machining functions.

4.1 Lexical analysis

The major functionality of lexical analysis is to merge a sequence of characters from the input NC program into sequence of words, which is a high-level representation unit, an example is shown in Figure 5. Meanwhile, in this step, all blank and comments within the program will be deleted. After lexical analysis, a symbol table with the same information but more systematic compared to the original character-based program will be built. During analysis, all character-based error will be checked, for example whether the unacceptable address letters has been used or not. In this paper, one dictionary has been designed in this step to store all the valid address letters.



```
N001 G91 G28 X0 Y0 Z0
N002 G00 X100 Y100
N003 S100. M03 M07
/N004 G90 G01 X150 Y150 F50.
N005 G17
N006 G02 X200 Y200 I50 J50
N007 #1001=300
N008 G01 X#1001
……

N110 M05 M09 M30
```

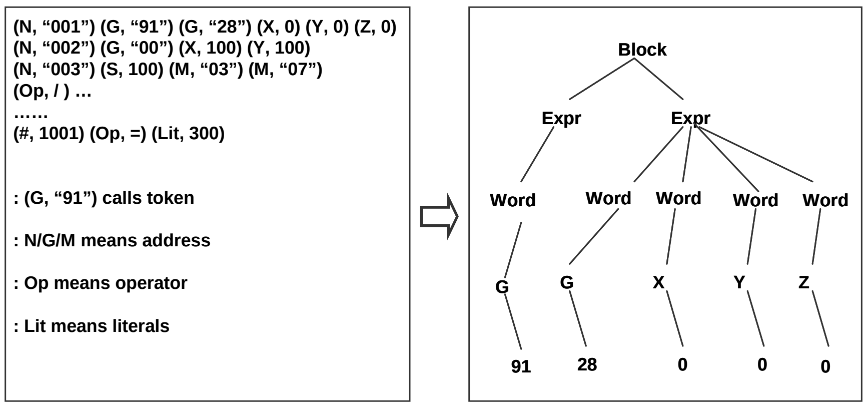
```
(N, “001”) (G, “91”) (G, “28”) (X, 0) (Y, 0) (Z, 0)
(N, “002”) (G, “00”) (X, 100) (Y, 100)
(N, “003”) (S, 100) (M, “03”) (M, “07”)
(Op, / ) …
……
(#, 1001) (Op, =) (Lit, 300)

: (G, “91”) calls token

: N/G/M means address

: Op means operator

: Lit means literals
```

Figure 5: Example of lexical analysis

4.2 Syntax and semantic analysis

Syntax analysis is to determine if a sequence of words within a block is syntactically correct, it is also called intra-block check. It includes the range checking of the data portion of a word and the parameters format checking.

Semantic analysis checks the major inter-block error, which means to make sure whether the logical relationship among several blocks of NC program is correct or not, for example, the same group G/M word cannot appear more than once in a block; block/word sequence should be subject to the G/M word execution order table, spindle should be turned on before any cutting motion starts, etc. Figure 6 shows an example of syntax and semantic analysis.



```
(N, “001”) (G, “91”) (G, “28”) (X, 0) (Y, 0) (Z, 0)
(N, “002”) (G, “00”) (X, 100) (Y, 100)
(N, “003”) (S, 100) (M, “03”) (M, “07”)
(Op, / ) …
……
(#, 1001) (Op, =) (Lit, 300)


: (G, “91”) calls token

: N/G/M means address

: Op means operator

: Lit means literals
```

Figure 6: Example of syntax and semantic analysis.

In order to design NCSD for syntax and semantic analysis, three kinds of cases of NC blocks should be analyzed:

Case 1: Extract data expression in each word of NC program. For example, “X [1+2*3-4/5], Zsin[30], #1=2.0 F #1” should be correctly decoded as “X6.2, Z0.5, F2.0”.

Case 2: Check syntax relation of words within each block. For example, within block "G17 G02 X10 Y20 I-10 F15 S100 M03", Z axis value should not appear since in XY circular interpolator plane only X&Y axes are allowed, also F word and S word are mandatory or at least should appear in advance in order to starting G02 circular motion.

Case 3: Check syntax relation among several blocks of NC program. For example, the following blocks:

"N0010 G91 G40

N0020 S100 M03

N0030 G01 G53 X20 F15"are subject to the G53 and G01 syntax rules, G91 in the first block "N0010 G91 G40" is correct, however an error will be found in block "N0030 …" because G53 does not allow G91 mode, which was given in block "N0010…".

Aiming at processing above three cases, a systematic and theoretical representation of the NC program syntax rules, Extended Backus Naur Form (EBNF) was chosen.

4.3 Extended Backus Naur Form

EBNF is a syntactic metalanguage to describe a computer programming language formally, which was developed by John Backus and Naur in 1960 to describe the syntax of the Algol 60 language. Since then, it has become a very important tool to represent the syntax of a language in computer science. An international standard for EBNF, ISO/IEC 14977 was also published in 1997. Following is a general definition of EBNF representation:

- Syntax consists of one or more syntax rules;
- A syntax rule is represented as an equation, left symbol of the equal sign is so-called non-terminal symbol while right symbols are a list consisting of either non-terminal symbol or terminal symbol;
- Non-terminal symbol appears at least once at the left of equal sign;
- Terms with quote symbol as header and ender are terminal symbol;
- Interpreting an expression is to apply one or more of the syntax rules.

NC language can be considered as one type of simple computer programming language; therefore it’s quite reasonable to use EBNF to represent the NC program syntax. Based on that, the structure of NCSD can be systematically defined.

4.3.1 Syntax representation using EBNF

The fundamental NC program syntax rules using EBNF are given in Figure 7. Based on these EBNF representations, case 1 mentioned in section 4.2 can be easily solved. Figure 8 shows an example of how to interpret word “X [1+2*3-4/5]” using these EBNF representations, as shown in this figure, two stacks (first in last out mechanism) are used: value stack and operator stack. The operator stack is subject to a rule: the priority of each item is always in a decreasing order while the execution with highest priority always happens first. Following is the ordered operation list of interpreting:

Production rules (portion for real value reading EBNF)

1. Real_value=real_number|expression|parameter_value|unary_combo;

2. Expression=’[‘+real_value+{binary_operation+real_value}+’]’;

    Binary_operation=’**’|’/’|’MOD’|’*’|’AND’|’XOR’|’-‘|’OR’|’+’;

3. Parameter_value=’#’+integer_number;

4. Unary_combo=ordinary_unary_combo|arc_tangent_combo;

5. Ordinary_unary_combo=ordinary_unary_operation+expression;

    Ordinary_unary_operation=’abs’|’acos’|’asin’|’cos’|’exp’|’sin’|’ln’|’sqrt’|’tan’;

6. Arc_tangent_combo=’atan’+expression+’/’+expression;

7. Real_number=[‘+’|’-‘]+((digit+{digit}+[‘.’]+{digit})|(’.’+digit+{digit}));

8. Integer_number=digit+{digit};

9. Digit=0|1|2|3|4|5|6|7|8|9;

Meta-identifier in the syntax EBNF

= The symbol on the left is equivalent to the expression on the right

+ followed by

| or

[] zero or one of the expression inside square brackets may occur

{} zero or many of the expression inside curly braces may occur

() exactly one of the expression inside parentheses must occur

; end of each rule

‘ first-quote-symbol

- except

Figure 7: Fundamental EBNF representation of NC language.

1) Apply first EBNF rule defined in Figure 7, read the symbol ‘[‘ and push it into the operator stack;

2) Apply EBNF rule 2,1,7,9 in turn, read value ‘1’ and symbol ‘+’, and then push them into value stack and operator stack respectively;

3) Continue to apply rule 1,7,9 twice, read ‘2’/‘3’ and ‘*’/‘-‘ separately, and push them into corresponding stacks;

4) Since operator ‘*‘ on the top of the operator stack has higher priority than its previous one ‘+’ and higher than the current one ‘-‘, current value 3’ is used to execute multiplication with ‘2’ popped up from the value stack. The result is pushed back into the value stack, while current operator ‘-‘ is pushed into the operator stack;

5) Continue the quite similar operation as above mentioned until the operator stack is empty and the last value ‘6.2’ is popped up from the value stack;

6) The last value ‘6.2’ is the final result. Therefore, “X [1+2*3-4/5” is interpreted as “X6.2”.

Example: interpreting “X [1+2*3-4/5]”

Operations	**Value stack**	**Operator stack**
Apply rule 1; read ‘[‘		[

Apply rule 2-1-7-9; read 1 and ‘+’	1	[+
Apply rule 1-7-9; read 2 and ‘*’	1 2	[+
Apply rule 1-7-9; read 3 and ‘-‘	1 2 3	[+ * -
Execute 2*3 because ‘+’<‘*’>’-‘	1 6	[+ -
Execute 1+6 because ‘+’>’-‘	7	[-
Apply rule 1-7-9; read 4 and ‘/’	7 4	[- /
Apply rule 1-7-9; read 5 and ‘]’	7 4 5	[- /]
Execute 4/5 because ‘-‘<’/’>’]’	7 0.8	[-]
Execute 7-0.8 because ‘-‘>’]’	6.2	[ ]
Apply rule 2-1; pop	6.2	

Figure 8: Example of interpreting by using EBNF.

For case 2&3, Figure 9 gives the EBNF representation of a general NC block while Figure 10 shows partial syntax EBNF representation of each G/M/F/S/T word in a group manner.

B.1 Block syntax EBNF:

```
Block = Group1_Expr | Group0_Expr1 | Other_Expr;
Group1_Expr = G80_Expr | Other_Group1;
Group0_Expr1 = G92_Expr | Other_Group0 ;
Other_Expr = M_Expr | S_Expr | F_Expr | T_Expr ;
G80_Expr = ‘g80’| G80_MODAL | (‘g80’ + Group0_Expr2);
Other_Group1 = G00_Expr | G01_Expr | G02_Expr |G03_Expr | G81_Expr | G82_Expr
               |G83_Expr | G84_Expr | G85_Expr|G86_Expr | G87_Expr | G88_Expr
               |G89_Expr ;
Group0_Expr2 = G10_Expr|G28_Expr|G30_Expr|G92_Expr;
Other_Group0 = G04_Expr|G53_Expr
```

Figure 9: EBNF representation of general NC block.

Let’s use a NC blocks example shown before in section 4.2 to explain how these EBNF rules are being applied. For each block of the following example:

```
“N0010  G91 G40
 N0020  S100 M03
 N0030  G01 G53 X20 F15“
```

EBNF rules B.1 in Figure 9 will be firstly applied, and then the detailed word EBNF rules in Figure 10 will be applied. Now for the first block, assuming the default modal is G01, the rule 1,2 and 6 of B.1 will take effective, then in rule 6 of B.1, rule B.3 is applied further and a correct check result will be returned. The similar procedure will be done for the second block. Then for the third block, as rule B.3 is applied, rule 4 of B.2 will be applied too, which is (G53_Expr=‘g53’ + ABS_Mode + G40_Expr + ‘g00’|’g01’). In this rule, ‘ABS_Mode’ (‘g90’) is required when ‘g53’ is given with ‘g01’, however, the third block does not satisfy this rule because ‘INC_Mode’ (‘g91’) is effective (this is given in the first block); therefore a syntax error is found.

4.3.2 Syntax dictionary implementation using TCL

The above mentioned EBNF-based syntax of NC program is programmed using TCL (tool command language), one kind of script language which works in an interpretive execution manner instead of compiling way. As all the NC program syntax is represented using TCL as TCL procedures, it will be loaded to work as the syntax dictionary of NCSD in the proposed NCPP.

B.2 G code group0 syntax EBNF

```
  G04_Expr =  ‘g04’ + ‘p’ + real_value;
  G10_Expr =  ‘g10’ + ‘l2’ + ‘p’ + (1-6) + [ Axis_Expr ];
  G28_Expr =  ‘g28’ + [ Axis_Expr] ;
  G53_Expr =  ‘g53’ + ABS_Mode + G40_Expr + ‘g00’|’g01’ ;
```

B.3 G code group1 syntax EBNF

```
  G01_Expr=[Coord_Expr] + [Feed_Mode] + [Distance_Mode]+ [ Unit_Input ] +
                [ Plane_Selection ] + ( ‘g01’ |G01_MODAL ) + Axis_Expr + F_Expr +
                ST_Expr + [ Coolant_On ] + [ G40_Expr ] + [ G53_Expr ];
  ……
```

B.13 M code syntax EBNF

```
  M_Expr = ( [ Spindle_Oper ] + [ Coolant_Oper ] + [ Change_Tool | ] ) | Prog_Oper;
- Spindle operation syntax EBNF
  Spindle_Oper = Start_Spindle | Stop_Spindle ;
  Start_Spindle = CW_Spindle | CCW_Spindle ;
  CW_Spindle = ‘m03’ ;
  CCW_Spindle = ‘m04’ ;
  Stop_Spindle = ‘m05’ ;
  ……
```

B.14 F code syntax EBNF

```
  F_Expr =  (‘ f ’ + real_value) | F_MODAL
  ……
```

Figure 10: EBNF representation of G/M word.

4.4 Processing engine design

4.4.1 Conceptual model of processing engine

With the implementation of NCSD in TCL procedures, a conceptual model of NCPP processing engine is proposed as shown in Figure 11. An event generator and an embedded TCL interpreter are involved to handle each input NC block in terms of the loaded syntax dictionary of NCSD. The event generator triggers the TCL interpreter by extracting words within a NC block according to priority. For each word’s syntax, a corresponding TCL procedure defined in syntax dictionary will be called by the TCL interpreter. The Syntax and semantic analysis will be done during this process; canonical machining functions will be generated too. The dictionary generator in this figure is used to generate the syntax dictionary from NC program syntax EBNF representation whenever a new NC specification is given.

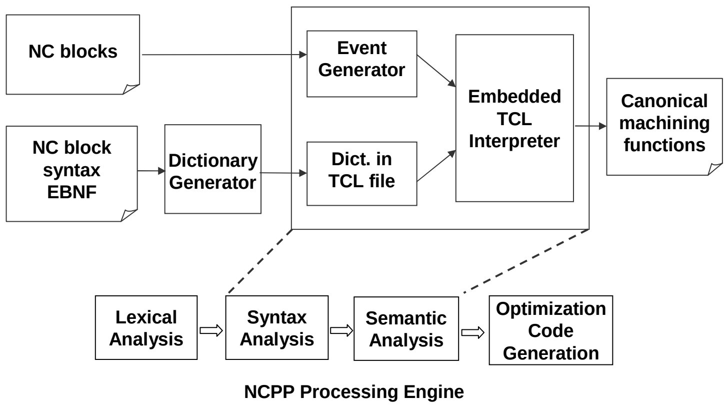


Figure 11: Conceptual model of NCPP processing engine.

4.4.2 Processing flow

Figure 12 shows the complete processing flow of proposed NCPP using one NC block example:

1) Initialization of the environment, which involves initializing parameters of NCPP (such as tool information, tool offset, coordination origin etc), loading the NCSD and starting the embedded TCL interpreter;
2) Suppose one block " N10 G94 G01 G90 X2 Y4 F5 M03 " is ready to process;
3) NCPP processing engine search for the WORD with highest priority within this block in terms of the priority rules table. In this case first is word G01 and second is M03. As shown in the figure, priority rules table gives the definition of priority of words as they explicitly appear in the same block. Basically, group1 of G word has the highest priority, followed by group0 and other groups. For example, if G01 (group1) and G92 (group0) both given in a block, G01 should be taken care first. In the same sense, G words precede F, M and T words. In addition, modal information has the same effect, which means that the G modal word from higher priority group takes effect as there is no G word explicitly appearing in the block.
4) Event generator of processing engine triggers the TCL interpreter to call and execute the corresponding TCL procedures in NCSD. Basically it will always call the procedure that is corresponding to EBNF rule B.1 first. (Rule B.1 represents the syntax of a general NC block). This procedure will indirectly call and execute the TCL procedure for G01 Syntax, while G01

procedure will further call G94 syntax procedure and G90 syntax procedure and so on. Then the TCL procedure of M03 syntax will be called. During the calling of each procedure, the corresponding syntax check of each word and the whole block will be done. The TCL interpreter will return the checking result either the ok or error message to the processing engine.

5) As the NCPP processing engine get the syntax check result, it will go to error handling if there is error, otherwise the current block will be further translated into the corresponding canonical machining functions.
6) Repeat step 3 - 5 until all blocks have been processed.

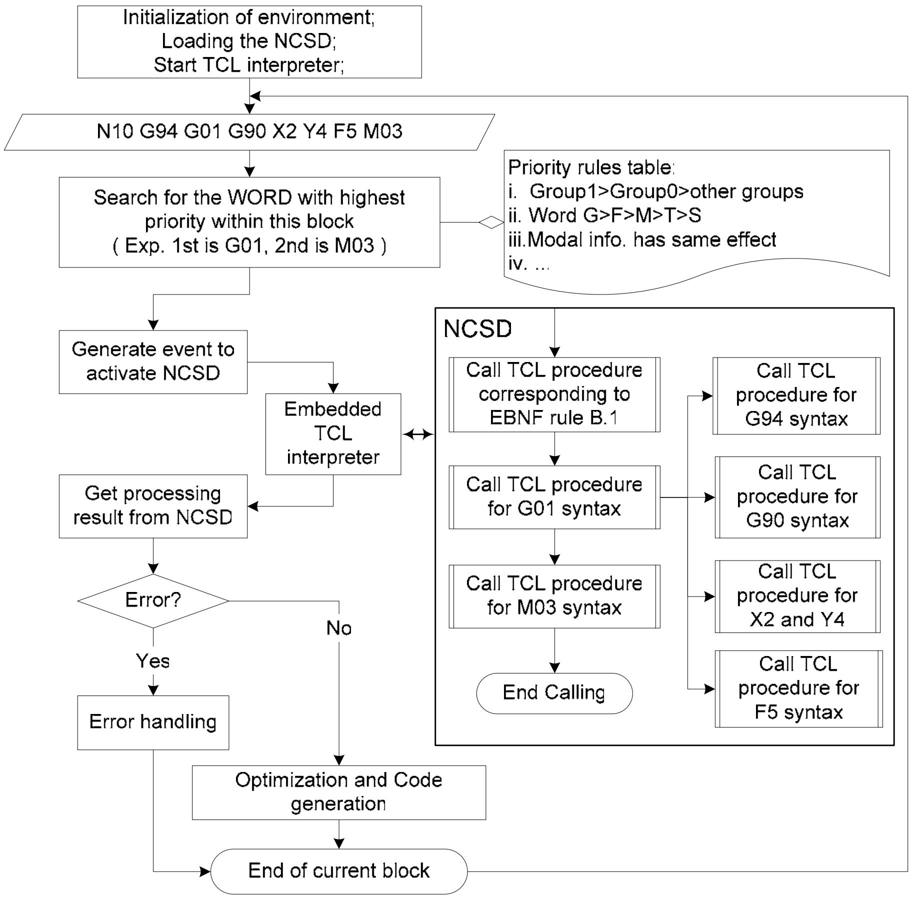

Initialization of environment;
Loading the NCSD;
Start TCL interpreter;
N10 G94 G01 G90 X2 Y4 F5 M03
Search for the WORD with highest priority within this block (Exp. 1st is G01, 2nd is M03)
Priority rules table:
i. Group1>Group0>other groups
ii. Word G>F>M>T>S
iii.Modal info. has same effect
iv. ...
Generate event to activate NCSD
Embedded TCL interpreter
Get processing result from NCSD
Error?
Yes
No
Error handling
Optimization and Code generation
End of current block
NCSD
Call TCL procedure corresponding to EBNF rule B.1
Call TCL procedure for G01 syntax
Call TCL procedure for M03 syntax
End Calling
Call TCL procedure for G94 syntax
Call TCL procedure for G90 syntax
Call TCL procedure for X2 and Y4
Call TCL procedure for F5 syntax

Figure 12: Processing flow of proposed NCPP engine.

5 PROTOTYPE SYSTEM IMPLEMENTATION

So far, a prototype system of proposed NCPP has been implemented using C and TCL. This prototype was developed based on an existing RS274NGC interpreter written by Thomas Kramer from NIST in 2000. The current version realized almost all the functionality proposed in this paper, while the dictionary generator of NCSD is still under developing. By using the standard RS274 and Fanuc specification NC program as input, the result of the developed prototype system shows that a successful design was obtained. Figure 13 shows a NC program example with two potential syntax errors as shown in italic and bold fonts. After being processed by proposed NCPP, two syntax errors are both detected out with error messages, as those shown in the figure 14 in italic and bold fonts. The corresponding canonical machining functions of the given NC program are generated too.

```
N0010 G40 G17 G94 G90
N0020 G91 G28 Z0.0
N0030 ~~S100 M03~~                (Spindle not rotate!)
N0040 G0 G90 X0.0 Y0.0
N0050 G43 Z10.H04 M08
N0060 G1 Z-2. F35.1
N0070 X-10. Y30.
N0080 G3 X-40. Y70. ~~I-41.667 J0~~     (G03 format error!)
N0090 G1 X250. Y36.6667
……
N1040 G0 X0.0 Y0.0 Z10.
N1050 M05
N1060 M02
%
```

Figure 13: A NC program example with syntax errors.

6 CONCLUSION

An intelligent NCPP is proposed for the CNC system of machine tool. It has separated NCSD and processing engine. NCSD varies in terms of different NC program format, while the engine is fixed. Based on this new structure, it is easy to adjust the CNC system to adapt to various NC program format by only updating the corresponding NCSD in NCPP. In this paper, the NCSD has been designed by using EBNF and implemented as TCL procedures. Meanwhile, the processing engine with embedded TCL interpreter has been built from a compiler's point of view. So far, a NCPP prototype system has already been built to evaluate the validity of the proposed design. In the future, in order to facilitate the preparation of the data to specify certain NC program specifications, it maybe useful to have user-friendly specification dictionary generator which can generate the dictionary in an efficient manner. This type of software may be referred to as NCSD generator.

```
 N0010  COMMENT("interpreter: feed mode set to units per minute")
 N0010  SELECT_PLANE(CANON_PLANE_XY)
 N0010  COMMENT("interpreter: cutter radius compensation off")
 N0020  COMMENT("interpreter: distance mode changed to incremental")
 N0020  STRAIGHT_TRAVERSE(0.0000, 0.0000, 0.0000)
 N0020  STRAIGHT_TRAVERSE(0.0000, 0.0000, 0.0000)
 N0040  COMMENT("interpreter: distance mode changed to absolute")
 N0040  STRAIGHT_TRAVERSE(0.0000, 0.0000, 0.0000)
 N0050  FLOOD_ON()
 N0050  USE_TOOL_LENGTH_OFFSET(0.0000)
N0050  STRAIGHT_TRAVERSE(0.0000, 0.0000, 10.0000)
TCL->MSG: missing spindle operation with g01 syntax
 N0060  SET_FEED_RATE(35.1000)
 N0060  STRAIGHT_FEED(0.0000, 0.0000, -2.0000)
N0070  STRAIGHT_FEED(-10.0000, 30.0000, -2.0000)
TCL->MSG: R i j k words all missing for arc
TCL->MSG: N0080 G3 X-40. Y70.
N0090  STRAIGHT_FEED(250.0000, 36.6667, -2.0000)
……
 N1040  STRAIGHT_TRAVERSE(0.0000, 0.0000, 10.0000)
 N1050  STOP_SPINDLE_TURNING()
 N1060  SET_ORIGIN_OFFSETS(0.0000, 0.0000, 0.0000)
 N1060  STOP_SPINDLE_TURNING()
```

```
N1060  FLOOD_OFF()
N1060  PROGRAM_END()
```

Figure 14: Processing result of the NC program with errors.

ACKNOWLEDGEMENT

The authors wish to express their sincere appreciation for the generous support from Mori Seiki Corporation which makes this research possible. We also owe our thanks to the work of Dr. Thomas Kramer from NIST, whose work laid a great foundation for this research project.

REFERENCES

[1] G.W.Vickors, M.H.Ly, R.G.Oetter, Numerically Controlled Machine Tools, Ellis Horwood, 1990.

[2] Peter Smid, CNC Programming Handbook, Industrial Press, 2000.

[3] Karen A. Lemone, Fundamentals of Compilers---An Introduction to Computer Language Translation, Prentice Hall, New Jersey, 1992.

[4] Thomas R. Kramer,The NIST RS274/NGC Interpreter - Version 3, ISD of NIST, 2000.

[5] Frederick M. Proctor, Canonical Machining Functions, ISD of NIST, 1997

[6] Ronald Mak, Writing Compilers and Interpreters---An Applied Approach, 1991.

[7] ISO/IEC 14977:1996(E) The standard of Extended BNF,1997.

[8] John K.Ousterhout, Tcl and the Tk Toolkit, Addison Wesley, Reading, 1994.