The NIST RS274/VGER Interpreter

Thomas R. Kramer
Frederick Proctor

Intelligent Systems Division
National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce
Gaithersburg, Maryland 20899

NISTIR 5754
November 9, 1995

Disclaimer

No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied.

Acknowledgements

Partial funding for the work described in this paper was provided to Catholic University by the National Institute of Standards and Technology under cooperative agreement Number 70NANB2H1213.

CONTENTS

1 Introduction

The NIST “RS274/VGER interpreter” is a software system which reads numerical control code in the “NGC” dialect of the RS274 numerical control language and produces calls to a set of canonical machining functions. The output of the interpreter can be used to drive a 5-axis machining center. This report describes the RS274/VGER interpreter.

1.1 Background

1.1.1 Enhanced Machine Controller Project

The Intelligent Systems Division (ISD) of the National Institute of Standards and Technology (NIST) is carrying out an Enhanced Machine Controller (EMC) project. The primary objective of the project is to build a testbed for evaluating application programming interface standards for open-architecture machine controllers. A secondary objective is to demonstrate implementations of the Next Generation Controller (NGC) architecture.

1.1.2 Numerical Control Programming Language RS274

RS274 is a programming language for numerically controlled (NC) machine tools, which has been used for many years. The most recent standard version of RS274 is RS274-D, which was completed in 1979. It is described in the document “EIA Standard EIA-274-D” by the Electronic Industries Association [EIA]. Most NC machine tools can be run using programs written in RS274. Implementations of the language differ from machine to machine, however, and a program that runs on one machine probably will not run on one from a different maker.

1.1.3 The RS274/NGC Language

The NGC architecture has many independent parts, one of which is a specification for the RS274/NGC language, a numerical control code language for machining and turning centers. The specification was originally given in an August 24, 1992 report “RS274/NGC for the LOW END CONTROLLER - First Draft” [Allen Bradley] prepared by the Allen Bradley company. A second draft of that document was released in August 1994 by the National Center for Manufacturing Sciences under the name “The Next Generation Controller Part Programming Functional Specification (RS-274/NGC)” [NCMS]. All references in this report are to the second draft. The term “the manual” in this report always means [NCMS]. The RS274/NGC language has many capabilities beyond those of RS274-D.

1.1.4 Previous Work at NIST

As part of ISD assistance to the program which developed the NGC architecture, ISD prepared a report “NIST Support to the Next Generation Controller Program: 1991 Final Technical Report,” [Albus] containing a variety of suggestions. Appendix C to that report proposed three sets of commands for 3-axis machining, one set for each of three proposed hierarchical control levels. The suite proposed for the lowest (primitive) control level was implemented in 1993 by the EMC project as a set of functions in the C programming language. This suite, known in the EMC project and in this report as the “canonical machining functions,” was upgraded in 1994 for 4-axis machining and has now been revised to be suitable for 5-axis machining.

Also in 1993, the authors developed a software system in the C language for reading machining commands in the RS274/NGC language and outputting canonical machining functions. This was called “the RS274/NGC interpreter.” A report, “The NIST RS274/NGC Interpreter, Version 1”

[Kramer1] was published in April 1994 describing that interpreter.

In 1994, the EMC project, in collaboration with the General Motors Company (GM), undertook to retrofit a 4-axis Kearney and Trecker 800 machine with an EMC controller. The retrofit was successfully completed in 1995. For this project NIST built both Version 2 of the RS274/NGC interpreter and an RS274KT interpreter, which interprets programs written in the K&T dialect of RS274. These two interpreters were written in the C++ programming language. Reports were written describing the RS274KT interpreter [Kramer2] and version 2 of the RS274/NGC interpreter [Kramer 3]. In addition, a report about the 4-axis canonical machining functions was written [Proctor].

1.1.5 Current Work at NIST

The EMC project is currently collaborating with several industrial partners in an open-architecture machine tool controller project known as VGER (which is a name, not an acronym). This project is retrofitting an SNK 5-axis machine tool with the open architecture controller being developed. The SNK machine has the usual X, Y, and Z axes, plus an A axis (parallel to the X axis) and a C axis (parallel to the Z axis). The A axis is the axis of a barrel-shaped part of the machine which holds the spindle mechanism. The C axis is the axis of a rotary table to which the workpiece is fixtured.

NIST is providing the RS274 interpreter for this project. It is intended to be able to interpret some existing programs for the SNK machine which were written for its former Fanuc controller, the language for which is described in [Fanuc]. The NGC and Fanuc dialects of RS274 are almost identical, but there are minor differences and points where [Fanuc] is consistent with [NCMS] but has additional requirements. Thus, the RS274/VGER interpreter takes Fanuc flavored RS274/NGC code as input. This language is called the RS274/VGER language in this report.

1.2 Overview of the RS274/VGER Language

This section gives an overview of RS274/VGER code. Further details of the meaning of RS274/VGER code are given in later sections of this report.

1.2.1 Lines, Blocks, Commands, and Words

The RS274/VGER language is based on lines of code. Each line (also called a “block”) may include commands to a machine tool to do several different things. A line is terminated by a carriage return or line feed. Lines of code may be collected in a file to make a program.

A typical line of code consists of an optional line number at the beginning followed by one or more “words,” possibly interspersed with comments. In this report, a word consists of a letter followed by a number or an expression that can be evaluated to a number. A word may either give a command or provide an argument to a command. For example, “G1 X3” is a valid line of code with two words. “G1” is a command meaning “move in a straight line at the programmed feed rate,” and “X3” provides an argument value (the value of X should be 3 at the end of the move) to the command. Most RS274/VGER commands start with either G or M (for miscellaneous). The words for these commands are called “G codes” and “M codes.” In addition to words, lines of code may include other combinations of characters. The legal combinations of characters are presented in Appendix F.

1.2.2 Commands and Machine Modes

In RS274/VGER, many commands cause the machine to change from one mode to another, and the mode stays active until some other command changes it implicitly or explicitly [NCMS, pages 71 - 73]. Such commands are called “modal”. If coolant is turned on, it stays on until it is explicitly turned off. The G codes for motion are also modal. If a G1 (straight move) command is given on one line, it will be executed again on the next line unless a command is given specifying a different motion (or some other command which implicitly cancels G1 is given).

“Non-modal” codes have effect only on the lines on which they occur. For example, G4 (dwell) is non-modal.

1.2.3 Modal Groups

Modal commands are arranged in sets called “modal groups”, and only one member of a modal group may be in force at any given time. In general, a modal group contains commands for which it is logically impossible for two members to be in effect at the same time — like measure in inches vs. measure in millimeters. A machine tool may be in many modes at the same time, with one mode from each modal group being in effect. The modal groups used in the interpreter are shown in Table 1.

For several modal groups, it is usual to provide that when the machine is ready to accept commands, one member of the group must be in effect. Controller manufacturers must set default values for those modal groups. When a machine is turned on or otherwise re-initialized, the default values are automatically in effect.

The modal groups for G codes are:

group 1 = {G0, G1, G2, G3, G38, G80, G81, G82, G83, G84, G85, G86, G87, G88, G89} - motion
group 2 = {G17, G18, G19} - plane selection
group 3 = {G90, G91} - distance mode
group 5 = {G93, G94} - spindle speed mode
group 6 = {G20, G21} - units
group 7 = {G40, G41, G42} - cutter diameter compensation
group 8 = {G43, G49} - tool length offset
group 10 = {G98, G99} - return mode in canned cycles
group 12 = {G54, G55, G56, G57, G58, G59, G59.1, G59.2, G59.3} - coordinate system selection

The modal groups for M codes are:

group 4 = {M0, M1, M2, M30, M60} - stopping
group 6 = {M6} - tool change
group 7 = {M3, M4, M5} - spindle turning
group 8 = {M7, M8, M9} - coolant
group 9 = {M48, M49} - feed and speed override switch bypass

Table 1. Modal Groups

In addition to the above modal groups, the interpreter has two groups for non-modal G codes:

group 0 = {G10, G28, G30, G92, G92.2} group 4 = {G4, G53}

1.2.4 Language Extensions

The RS274/VGER language includes three commands that are required to use common functions of some machining centers but are missing from the RS274/NGC language, as defined in [NCMS]. These deal with pallet shuttle and operator messages. Other extensions to the RS274/NGC language are discussed in [Kramer3].

Pallet Shuttle

The pallet shuttle function of a machining center causes two pallets to change places. In the RS274/VGER language, M30 causes a pallet shuttle and the end of a program, and M60 causes a pallet shuttle and program stop.

Messages for the Operator

In the interpreter, a comment is anything enclosed in parentheses. If a comment begins with "MSG," it is considered to be a message, and the MESSAGE canonical function is called rather than the COMMENT canonical function. The MESSAGE canonical function causes the message, which is the rest of what is in parentheses, to be displayed on the operator console.

1.3 Canonical Machining Functions

The EMC canonical machining functions called by the interpreter are given in Table 2. Four-axis versions of these commands are described in more detail in a separate report, "Canonical Functions for 4-Axis Machining" [Proctor]. That report includes additional canonical machining functions which are not used by the interpreter but may be in future versions. The names of the functions explain roughly what they do. The 5-axis canonical commands given here are a simple extension of the 4-axis commands. Wherever the 4-axis commands have arguments dealing with the B axis (a rotary axis parallel to the Y axis), the 5-axis commands have two corresponding arguments, one for the A axis and one for the C axis.

The canonical machining commands are atomic commands. Each command produces a single action. RS274 commands, on the other hand, include two types: those for which a single RS274 command corresponds exactly to a canonical command, and those for which a single RS274 command will be decomposed into several canonical commands (possibly dozens). Things like "move in a straight line" or "turn flood coolant on" are of the first type. Things like "turn all coolant off" or "run a peck drilling cycle" are of the second type.

The canonical commands used in the interpreter were devised with three objectives in mind. First, all the functionality of the SNK machine had to be covered by the commands; for any function the machine can perform, there has to be a way to tell it to do that function. Second, it was desired that it be possible to use readily available commercial motion control boards from various vendors to carry out those canonical commands which call for motion, with roughly a one-to-one correspondence between a canonical motion command and a command a commercial board recognizes. Third, it must be possible to interpret RS274/VGER commands into canonical commands.

Two sets of definitions for the canonical machining functions have been written, and either set can be linked into the interpreter. The first set is used for direct control of the machining center. Executing a function from this set causes a command message to be generated. When this command message is executed, the machine's actuators are activated. The second set is used for testing or for writing a command file that can be used later. Executing a function from the second set causes a line of text containing the command to be written to standard output or to a file.

Representation	SET_ORIGIN_OFFSETS (double x, double y, double z, double a, double c) USE_LENGTH_UNITS (CANON_UNITS units)
Free Space Motion	STRAIGHT_TRAVERSE (double x, double y, double z, int a_turn, double a_position, int c_turn, double c_position)
Machining Attributes	SELECT_PLANE (CANON_PLANE plane) SET_FEED_RATE (double rate) SET_FEED_REFERENCE (CANON_FEED_REFERENCE reference) START_SPEED_FEED_SYNCH() STOP_SPEED_FEED_SYNCH()
Machining Functions	ARC_FEED (double first_end, double second_end, double first_axis, double second_axis, int rotation, double axis_end_point, int a_turn, double a_position, int c_turn, double c_position) DWELL (double seconds) STRAIGHT_FEED (double x, double y, double z, int a_turn, double a_position, int c_turn, double c_position)
Probe Functions	STRAIGHT_ PROBE (double x, double y, double z, int a_turn, double a_position, int c_turn, double c_position)
Spindle Functions	SET_SPINDLE_SPEED (double r) START_SPINDLE_CLOCKWISE () START_SPINDLE_COUNTERCLOCKWISE () STOP_SPINDLE_TURNING () ORIENT_SPINDLE (double orientation, CANON_DIRECTION direction)
Tool Functions	CHANGE_TOOL (int slot) SELECT_TOOL (int i) USE_TOOL_LENGTH_OFFSET (double offset)
Miscellaneous Functions	COMMENT (char * s) DISABLE_FEED_OVERRIDE() DISABLE_SPEED_OVERRIDE() ENABLE_FEED_OVERRIDE() ENABLE_SPEED_OVERRIDE() FLOOD_OFF () FLOOD_ON () MESSAGE (char * s) MIST_OFF () MIST_ON () PALLET_SHUTTLE()
Program Functions	OPTIONAL_PROGRAM_STOP () PROGRAM_END () PROGRAM_STOP ()

Table 2. Canonical Machining Functions Called By Interpreter

Function arguments are written in ANSI C style. All functions return nothing.

2 Overview of the Interpreter

2.1 Major Characteristics

2.1.1 Modes of Use

The interpreter runs integrated with the EMC control system or as a stand-alone system. Most of the software is the same in the two cases. Software details are given in Appendix A. The reader may find it helpful to look at Figure 1 in Appendix A at this point.

2.1.1.1 Integrated with EMC Control System

In the EMC control system, the interpreter is used both to interpret NC programs (from files) and to interpret individual commands entered using the manual data input (MDI) capability of the control system. When running an NC program, the control system tells the interpreter when to read another line of code from the program and when to execute the last line that was read. When using MDI input, the controller sends the interpreter a line of code it gets from the user interface with a single command that tells the interpreter to read the line and execute the line.

The interpreter does not control machine action directly. Rather, the interpreter calls canonical functions which generate messages which are passed back to the control system, and the control system decides what to do with the messages. In normal operation the top level of the control system decides whether each message should be sent to its motion control subordinate or to its discrete I/O subordinate and sends the message at an appropriate time.

If an error occurs, the user is sent an error message. If an NC program is being translated, execution of the program stops at the line where the error occurred, and it is not possible to restart the program from that line. To use a program which causes an interpreter error, the program must be edited to remove the error, and the program must be restarted at the beginning. The user must determine if the partially cut workpiece can be saved or whether it must be scrapped, and must consider the effect of re-running the portion of the program before the error occurred in making this decision. Of course the user has the option of running a revised program which deletes the code that ran successfully at the beginning of the original program. Further details of error handling are given in Appendix B.1

2.1.1.2 Stand-alone

In stand-alone mode, the interpreter has two input modes: keyboard (interactive) mode and file (batch) mode. In the interactive mode, the user types lines of RS274/VGER code at the keyboard. In the batch mode, the interpreter reads lines of code from a file.

Only the set of canonical machining functions which prints text has been linked into the stand-alone interpreter, so the output is always text. The output is printed to the computer terminal by default, but may be redirected to a file. In general, an output file is useful only if input is taken from a file and errors do not occur during interpretation.

Error handling differs between the interactive mode and the batch mode. In interactive mode, the interpreter always continues after an error. In batch mode the user has an option when starting the interpreter of telling the interpreter to try to keep going after an error or having the interpreter stop interpreting if an error occurs (which is the default behavior).

The stand-alone mode is valuable because it allows a user to pre-test an NC program without

having to run it on the machine controller itself. Any computer for which the stand-alone interpreter can be compiled can be used to pre-test NC programs. Pre-tests are conclusive tests because the interpreter runs exactly the same way in the stand-alone mode as it does integrated with the control system.

Section 4.1 explains how to use the various options available in stand-alone mode.

2.1.2 How it Runs

Once initialized, the interpreter runs in various ways (depending on which of the modes of use just described is being used) but in all of them, the basic operation that gets repeated is a two-step process:

1. Get a line of RS274/VGER code and read it into memory, building an internal representation of the meaning of the entire line.
2. Call one or more canonical machining functions which will do what the line says to do.

The interpreter maintains a model of the machine while it is interpreting and uses the model in determining what canonical machining functions to call and what their arguments should be. Initialization of the model is performed when the interpreter starts up.

2.1.3 Speed

2.1.3.1 Stand-alone Speed

Using an input test file for machining a semicircular arc back and forth 1000 times in a row (for a total of 2000 lines), running on a SUN SPARCstation 2, the stand-alone interpreter wrote an output file in 4 seconds. A second file with 3600 small arcs lying on a helix took 11 seconds. A third file with 2000 straight line segments took 5 seconds. These three tests show a rate of 327 to 500 lines per second for the stand-alone interpreter on a Sun SPARCstation 2 computer. In recent tests on a Sun SPARCstation 20, the stand-alone interpreter ran about five times as fast as on a SPARC2. The program with 3600 arcs, for example, ran in 2 seconds.

Running on a 486 PC using the Lynx operating system, the stand-alone interpreter handled the same three test files in 14 seconds, 25 seconds, and 13 seconds, about a third as fast as on the Sun SPARC. The output from the two computers was byte-by-byte identical for the first two input programs. For the third program there were about 100 numbers which differed in the last decimal place, apparently because of different conventions for rounding used by the two computers when the digit after the last decimal place to be kept is a 5.

Tests with other types of programs show similar results. Thus, it is clear that using the stand-alone system for pre-testing NC programs takes little time.

2.1.3.2 Interpreter Speed in the Integrated System

When the interpreter is being used integrated with the EMC control system, there are relatively few situations in which a modern PC would be challenged, since the tool path prescribed by a line found in a typical program will rarely take less than a tenth of a second to cut, while interpreting that line will rarely take more than a hundredth of a second. Thus, for most NC programs, the interpreter speed will not be a significant factor.

There is one known class of program for which interpreter speed may be a significant factor. This class is those programs which include many consecutive short lines or arcs (say 0.1 millimeter

long each). This type of program is produced by many NC program post-processors for following complex contours approximately. At 1000 millimeters per minute feed rate (a realistic value), a 2000-line file should run in 12 seconds. Tests of this type of program on the integrated system have not yet been conducted, but the results just mentioned for the stand-alone interpreter running on a PC indicate that even without all the other tasks the integrated system must perform, this speed challenges the interpreter. The interpreter has not been optimized for speed. If it is decided more speed is required during execution, that may be accomplished by various types of pre-processing which are not difficult to implement.

2.2 Start-up

When the interpreter starts up, before accepting any input, it sets up a model that includes data about itself, data about the setup of the machine to be controlled, and data about the tools which are in the tool carousel of the machine. The interpreter's data about itself (such as whether tool radius compensation is on) has default values. The data about the machine and the tools is obtained in two different ways depending upon whether the interpreter is integrated or stand-alone.

In the integrated mode, machine setup data for the model is obtained by the interpreter from the (real) control system database. This data (such as axis positions) has been read from the sensors on the machine and represents the actual state of the machine. Also in this mode, the data about tools is obtained from the information that has been input by an operator.

In the stand-alone interpreter, machine setup data and tool data for the model is obtained (using exactly the same queries) from a stub version of the database of the control system which has default values for the data the interpreter needs. The default data does not necessarily represent the actual state of the machine or the tools in the carousel.

In the stand-alone interpreter, all the machine setup data and the tool data may be replaced by data from files designated by the user. In this way the stand-alone interpreter can be given data representing different conditions of the machine and tools; the data in the files may be actual or hypothetical. When the interpreter starts up in the stand-alone mode, the user is prompted to provide the name of a setup file and the name of a tool file. In both cases, if a name is provided, the data from the file is used. If a name is not provided, the data already loaded is used. The formats for setup files and tool files are described in Appendix G and Appendix H, respectively, with examples of both types of file.

Default setup data is shown in Table 3. The tool table indexes used in Table 3 correspond to slot numbers from the tool changer on the machine.

Default tool data is shown in Table 4. The default tool table is not intended to be useful for preparing or testing any real NC programs. Data has been placed in the default tool table so the interpreter can be used without having to prepare a tool table. The default tool table has the tool length offset and the tool diameter as zero for all slots except slots 1 and 2.

The values in the tool table are used as though they are in the units (inches or millimeters) currently in use; the tool table values are not adjusted automatically if units are changed.

Item	Setting	RS274/VGER code	Internal/External
axis_offset_a	0.0	G92.2	I
axis_offset_c	0.0	G92.2	I
axis_offset_x	0.0	G92.2	I
axis_offset_y	0.0	G92.2	I
axis_offset_z	0.0	G92.2	I
block delete switch	off	console switch	E
coordinate system number	1	G54 - G59.3	I
current_a	0.0		E
current_c	0.0		E
current_x	0.0		E
current_y	0.0		E
current_z	0.0		E
cutter radius compensation	off	G40	I
distance mode	absolute	G90	I
feed mode	units per minute	G94	I
feed override	enabled	M48	I
feed rate	15		I
flood coolant	off	M9	E
length units	millimeters	G21	I
mist coolant	off	G9	E
motion mode	80	G80	I
plane for arcs	XY plane	G17	E
origin_offset_a	0.0	G10 & G54 - G59.3	E
origin_offset_c	0.0	G10 & G54 - G59.3	E
origin_offset_x	0.0	G10 & G54 - G59.3	E
origin_offset_y	0.0	G10 & G54 - G59.3	E
origin_offset_z	0.0	G10 & G54 - G59.3	E
slot in use	1		E
slot selected	1		I
slot for length offset	1		I
slot for radius comp	1		I
speed_feed_mode	independent		I
speed override	enabled	M48	I
spindle speed	0.0		E
spindle turning	not turning	M5	E
tool length offset	0.0		I
traverse rate	100		E

Table 3. Default Setup Data for the Interpreter

In addition to the data shown here, the values of many parameters in the parameter table are set.

Slot	ID	Length	Diameter
1	1	2.0	1.0
2	2	1.0	0.2

Table 4. Default Tool Data for the Interpreter

All other slots have the id, length, and diameter set to zero.

2.3 Exiting

When using keyboard input, the interpreter exits only if it reads a line with the one word “quit”. Most variations of “quit” are valid, e.g., “Q uI t”. In any other mode, the interpreter exits when it reads a line with an M2 (program exit) or M30 (program exit with pallet shuttle) command on it. The rest of the line is executed before the interpreter exits. If there are more lines in the file following the line which causes a program exit, they are ignored.

As stated earlier, in some modes which use file input, the interpreter will also exit if a line is read that causes an error. It is an error for the file to come to an end without an M2 or M30 command.

3 Building a Stand-Alone Executable

On a SUN SPARCstation 2, an executable file for the stand-alone interpreter may be built from source code in about a minute, as described below. The same procedure should work on any computer running a Unix operating system and having the standard C++ libraries. On computers running other operating systems, compilation should be similarly easy, provided the standard C++ libraries are available.

To make an executable, eight source code files must be placed in the same directory along with the Makefile shown in Table 5 below. The source code files are:

canon.cc
canon.hh
rs274vger.cc
rs274vger.hh
nce_code.h
nce_err_sun.c
driver.cc
nml_emc.hh

The first two files are the function definitions and header file for the canonical machining functions. The version of canon.cc which prints function calls is used with the stand-alone interpreter. The second two are the function definitions and header file for the interpreter kernel and interface functions. The next two are the header file and definition file for error messages. The last two are the function definitions and header file needed for the stand-alone interpreter that are not needed in the integrated interpreter.

An executable file named “rs274vger” is built in the same directory by giving the command: **make rs274vger**.

In the Makefile, we are using the Gnu C++ compiler, “g++.” Any other C++ compiler may be substituted for g++.

```
canon.o: canon.cc canon.hh nml_emc.hh rs274vger.hh
        g++ -c -v -g -O canon.cc
rs274vger.o: rs274vger.cc canon.hh nml_emc.hh rs274vger.hh nce_code.h
        g++ -c -v -g -O rs274vger.cc
driver.o: driver.cc canon.hh nml_emc.hh rs274vger.hh
        g++ -c -v -g -O driver.cc
nce_err_sun.o: nce_err_sun.c
        g++ -c -v -g -O nce_err_sun.c

rs274vger: rs274vger.o canon.o driver.o nce_err_sun.o
        g++ -v -o rs274vger rs274vger.o canon.o driver.o nce_err_sun.o -lm
```

Table 5. Makefile for Interpreter

4 Using the Stand-Alone Interpreter

4.1 Invoking the Interpreter

As mentioned earlier, the stand-alone interpreter may be used with either keyboard input or file input. This section tells how to do that. The user must press the “return” key after each response; the return key presses are not printed here.

This section describes several input files. If any input file named by the user cannot be opened, the interpreter prints a message to that effect and quits.

This section refers to redirecting output. The methods described here work with both the SunOS and LynxOS and may be expected to work on other Unix-like systems. Other forms of redirection are possible.

4.1.1 Invocation with Keyboard Input

The interpreter is invoked with keyboard input by giving the command:

rs274vger

4.1.2 Invocation with NC File Input

4.1.2.1 Invocation to Stop After an Error

To use NC file input and stop if an error is encountered, invoke the interpreter with a command of the form:

rs274vger *input_filename*

where input_filename is the name of the NC input file. With this invocation, normal printed output from the interpreter (everything but error messages) appears on stdout, which is normally the terminal on which the command was invoked. Printed output may be usefully redirected to an output file by giving a command of the form:

rs274vger *input_filename* > *output_filename*

where output_filename is the name the output file should have. The interpreter will create this file if it does not exist; if it does exist, it will be overwritten.

4.1.2.2 Invocation to Continue After an Error

To use file input and attempt to continue if an error is encountered, invoke the interpreter with a command of the form:

rs274vger *input_filename* continue

As above, normal printed output from the interpreter (everything but error messages) appears on stdout. Error messages are printed to stderr, which is also normally the terminal from which the interpreter was invoked. Printed output (excluding error messages) may be redirected to an output file by giving a command of the form:

rs274vger *input_filename* continue > *output_filename*

If there are errors during interpretation, any input line which causes an error will not be interpreted, and the output file may be incorrect on any line after the last line which was output before the first error occurred.

4.2 Tool and Setup Files

As soon as the interpreter is invoked by any of the methods just described, it prompts the user to enter the name of a tool file. The user may enter a tool file name, as follows:

`name of tool file =>` ***tool_file_name***

or the user may press only the return key, and the interpreter will use the default tool data.

The interpreter next prompts the user for the name of a setup file. The user may enter a setup file name as follows:

`name of setup file =>` ***setup_file_name***

or the user may press only the return key, and the interpreter will use the default setup data.

The format of setup files and tool files is described in Appendix G and Appendix H, respectively.

4.3 Keyboard User Interface

The interpreter has a simple line-based user interface for when it is used with keyboard input. The normal pattern of use is a read-execute cycle with four steps:

1. The interpreter prints the prompt **`READ =>`**

2. The user enters a line of RS274/VGER code at the keyboard and hits the carriage return button.

3. The interpreter reads the line and prints the prompt **`EXEC <-`**

4. The user enters a semicolon and hits the carriage return button, and the line is interpreted.

Steps 3 and 4 have been included in the interface to give the user a chance to check the line just typed. If anything but a semicolon is entered in step 4, the line is not interpreted. This will protect the user who accidentally hits the return button during step 2.

The user interface provides no capability to edit ahead, undo, or anything else involving more than

the current line.

A transcript of a short session with the interpreter using keyboard input is shown in Appendix D.

5 INPUT

5.1 Overview

In general, allowable inputs are as described in [NCMS] or [Fanuc]. [EIA], the standard for RS274-D is used where the [NCMS] and [Fanuc] are silent, but [EIA] has something to say.

5.1.1 White Space

The manual says nothing about space characters or tab characters. [EIA, page 6, last line] allows spaces and tabs anywhere and provides that they should be "ignored by control." The interpreter allows spaces and tabs anywhere on a line of code and behaves the same as it would if they were not there. This makes some strange-looking input legal. The line "g0x +0. 12 34y 7" is equivalent to "g0 x+0.1234 y7", for example.

Blank lines are allowed in the input by the interpreter. They are ignored.

5.1.2 Case Sensitivity

The manual and [EIA] do not explicitly discuss character case. The interpreter assumes input is case insensitive, i.e., any letter may be in upper or lower case without changing the meaning of a line.

5.2 Input Lines

To make the specification of an allowable line of code precise, we have defined it in a production language (Wirth Syntax Notation) in Appendix F. The description here is intended to be consistent with the appendix. In order that the definition in the appendix not be unwieldy, many constraints imposed by the interpreter are omitted from that appendix. The list of error messages in Appendix E indicates all of the additional constraints.

5.2.1 Format of a Line

A permissible line of input consists of the following, in order, with the restriction that there is a maximum (currently 256) to the number of characters allowed on a line.

1. an optional block delete character, which is a slash / .
2. an optional line number.
3. any number of segments, where a segment is a word or a comment.
4. an end of line character.

5.2.2 Word

A word is a letter followed by a real_value.

A real_value is some collection of characters that can be processed to come up with a number. A real_value may be an explicit number (such as 341 or -0.8807), a parameter_value, an expression, or a unary operation value.

5.2.3 Number

The manual is not clear regarding what a valid number is. On page 2 it says "The programming

resolution of the axis word formats word length will be 8." The most likely interpretation of this is that a word representing an axis motion can have at most eight characters (presumably including one letter at the beginning and a decimal point in the middle), providing room for six digits. The table on pages 5 and 6 provides for up to 14 digits, but a sentence on page 5 says the table may not be valid, and "the control allows a maximum of 8 total digits."

[NCMS, page 2] says "Decimal Point Programming of Axis motion will be the standard." It does not say what Decimal Point Programming is, however. [EIA, page 3] does describe decimal point programming. It allows all unnecessary zeros to be included or suppressed and decimal points to be omitted if whole numbers are used.

The interpreter uses the following rules regarding numbers. In these rules a digit is a single character between 0 and 9.

- A number consists of (i) an optional plus or minus sign, followed by (ii) zero to many digits, followed, possibly, by (iii) one decimal point, followed by (iv) zero to many digits — provided that there is at least one digit somewhere in the number.
- There are two kinds of numbers: integers and decimals. An integer does not have a decimal point in it; a decimal does.
- Numbers may have any number of digits, subject to the limitation on line length.
- A non-zero number with no sign as the first character is assumed to be positive.

Notice that initial (before the decimal point and the first non-zero digit) and trailing (after the decimal point and the last non-zero digit) zeros are allowed but not required. A number written with initial or trailing zeros will have the same value when it is read as if the extra zeros were not there.

Numbers used for specific purposes in RS274/VGER are often restricted to some finite set of values or some to some range of values. In many uses, decimal numbers must be close to integers; this includes the values of indexes (for parameters and changer slot numbers, for example), M codes, and G codes multiplied by ten. In the interpreter, a decimal number which is supposed be close to an integer is considered close enough if it is within 0.0001 of an integer.

5.2.4 Line Number

A line number is the letter N followed by an integer (with no sign) between 0 and 99999 [NCMS, page 7].

5.2.5 Parameter_value

A parameter_value is the pound character # followed by a real_value which must evaluate to an integer between 1 and 5399. The integer is an index of a parameter table and the value of the parameter_value is whatever number is stored in the parameter table at that index.

The # character takes precedence over other operations, so that, for example, "#1+2" means the number found by adding 2 to the value of parameter 1, not the value found in parameter 3. This is implied but not explicit in the manual. Of course, #[1+2] does mean the value found in parameter 3. The # character may be repeated; for example ##2 means the value of the parameter whose index is the (integer) value of parameter 2.

5.2.6 Expressions and Binary Operations

An expression is a set of characters starting with a left bracket [and ending with a balancing right bracket]. In between the brackets are numbers, parameter_values, mathematical operations, and other expressions. An expression may be evaluated to produce a number. The interpreter does expression evaluation while it is reading, before executing anything in a block. An example of an expression is [1 + acos[0] - [#3 ** [4.0/2]]].

[NCMS section 3.5.1.1] discusses expression evaluation.

The manual provides for eight binary operations, and a ninth has been added to the interpreter. There are four basic mathematical operations: addition (+), subtraction (-), multiplication (*), and division (/). There are three logical operations: non-exclusive or (OR), exclusive or (XOR), and logical and (AND). The eighth operation is the modulus operation (MOD). The added ninth operation is the “power” operation (**) of raising the number on the left of the operation to the power on the right; this is needed for many basic machining calculations.

The binary operations are divided into two groups. The first group is: multiplication, division, modulus, and power. The second group is: addition, subtraction, logical non-exclusive or, logical exclusive or, and logical and. If operations are strung together (for example in the expression [2.0 / 3 * 1.5 - 5.5 / 11.0]), operations in the first group are to be performed before operations in the second group. If an expression contains more than one operation from the same group (such as the first / and * in the example), the operation on the left is performed first. Thus, the example is equivalent to: [((2.0 / 3) * 1.5) - (5.5 / 11.0)] , which simplifies to [1.0 - 0.5] , which is 0.5.

The logical operations and modulus are apparently to be performed on any real numbers, not just on integers or on some other data type. In the interpreter, the number zero is equivalent to logical false, and any non-zero number is equivalent to logical true.

5.2.7 Unary Operation

A unary operation value is either “ATAN” followed by one expression divided by another expression (for example “ATAN[2]/[1+3]”) or any other unary operation name followed by an expression (for example “SIN[90]”). The unary operations are: ABS (absolute value), ACOS (arc cosine), ASIN (arc sine), ATAN (arc tangent), COS (cosine), EXP (e raised to the given power), FIX (round down), FUP (round up), LN (natural logarithm), ROUND (round to the nearest whole number), SIN (sine), SQRT (square root), and TAN (tangent). The arguments to unary operations which take angle measures, such as sine, are in degrees.

The meaning of FIX for negative numbers is not clear in the manual. The interpreter rounds down towards zero, so that FIX[-2.8] is -2, for example. A second possible meaning of “round down,” which is not used in the interpreter, is “round towards the less positive or more negative”. A similar ambiguity exists for FUP. The interpreter rounds up away from zero, so FUP[-2.8] is -3, for example.

5.3 Word Repeats

The manual does not specify explicitly whether two words starting with the same letter can occur in the same line, but portions of the manual ([NCMS], page 73], for example) imply clearly that this is legal. [EIA] has an explicit statement (item 3.12, page 5) that “words (apparently meaning words with the same initial letter) shall not be repeated in a block.” The interpreter uses the following rules:

- A line may have any number of G words, but two G words from the same modal group (or from either non-modal group 0 or group 4) may not appear on the same line.
- A line may have zero to four M words. Two M words from the same modal group may not appear on the same line.
- For all other legal letters, a line may have only one word beginning with that letter.

5.4 Word order

The manual does not specify word order explicitly or implicitly. The interpreter allows words starting with any letter except N (which denotes a line number and must be first) to occur in any order. Execution of the line will be the same regardless of the order.

5.5 Measurement Units

5.5.1 Linear units

The manual specifies that either “inch” or “metric” units may be used by programming G20 for inch or G21 for metric [NCMS, page 15]. The default is up to the system builder. One or the other is automatically to be chosen when the system starts up. The stand-alone interpreter starts up using millimeters, as shown in Table 3.

The manual uses the term “metric” frequently ([NCMS, pages 2 and 15] for example). The use of the term implies clearly that in many cases it is meant to denote a linear measurement. Where “metric” is used as a linear measurement, [NCMS] does not say directly what metric unit is intended, although “millimeters” is used in at least one table [NCMS, page 35]. The interpreter uses millimeters.

5.5.2 Angular units

[NCMS, pages 18, 19, 34] specifies using degrees for measuring angles, so the interpreter assumes angular dimensions in the input are given in degrees.

5.6 Rotary Axes

The SNK machine being controlled has A and C rotary axes, as described earlier.

5.6.1 Coordinate Values for Rotary Axes

The coordinate values for the A and C axes in the input NC code may be any real number. Values for these axes are always given in degrees.

In the RS274/VGER language, the scales for the rotary axes are treated like wound-up linear axes. For example, if the original position of the A axis is 0.0, and the axis is rotated through two full turns counterclockwise, its new value is 720.0 (since there are 360 degrees in a full turn, and 2 x 360 = 720). This is somewhat awkward, since the machine configuration is unchanged by two full turns, and the axis looks like it is back to 0.0. Another awkwardness is that if the axis keeps turning in the same direction the value of its position keeps getting larger.

In some other dialects of RS274, [K&T] for example, the value of a rotary axis is also expressed as a real number, but the meaning is quite different. In this other method, the absolute value of the number represents a position between 0 and 359.999 where the axis must stop at the end of its motion, and the plus or minus sign in front of the number means to turn counterclockwise or clockwise. In this method, after two full turns starting from 0.0, the position is still 0.0.

In [NCMS] and [Fanuc] it is not possible to tell which of the two methods just described is intended. The method selected for RS274/VGER was selected because it is the method used in some existing NC programs which the interpreter was designed to handle.

5.6.2 Feed Rate for Rotary Axes

If a rotary axis moves when any of the X, Y, or Z axes moves, the rotary axis is slaved to the XYZ axis motion.

If one rotary axis moves and no other axis moves, the current feed rate is interpreted as meaning degrees per minute.

If two rotary axes move and none of the X, Y, or Z axes move, the move is treated as though the size of the move were the square root of the sum of the squares of the amounts of A and C axis moves. That is, the total amount of time the move will take is calculated by dividing that calculated size by the current feed rate, and the two axes are rotated simultaneously so that each moves at the constant angular rate which will make the axis take the right amount of time to complete its motion.

5.7 Messages and Comments

[NCMS, page 3] prescribes using parentheses to give comments. It does not specify whether nested comments are allowed or whether a line can end in the middle of a comment without being closed by a right parenthesis. The interpreter assumes comments can occur anywhere in a line but may not contain left parentheses and must be terminated by a right parenthesis before the end of the line, or the line is invalid and will not be executed.

In the interpreter, it is allowed to have more than one set of parentheses on a line (like) (this), but if this is done, only the last one will be processed as described below. All the others will be read and their format will be checked, but they will be ignored thereafter.

5.7.1 Messages

If “MSG,” appears after a left parenthesis before any other printing characters, the rest is of the characters before the right parenthesis are considered to be a message, and the “message” canonical function is called to deliver the message to the operator. Variants of “MSG,” which include white space and lower case characters are allowed.

5.7.2 Comments

If the characters inside parentheses are not a message, as just described, then everything inside the parentheses is treated as a comment, and the “comment” canonical function is called. Comments do not cause the machine to do anything.

5.8 Programs

By “program” we mean a sequence of lines of RS274/VGER NC code (at least one and perhaps as many as several thousand) that are intended to be executed one after another. The sequence of lines is normally kept in together in a file.

The stand-alone interpreter has no concept of a program when it is running with keyboard input; it only understands lines. When it runs with input from a file, it expects the file to be an NC program. The interpreter exits when a line with M2 or M30 is executed, since they mean end of program.

In the integrated interpreter, programs are recognized and handled by the user interface of the EMC control system.

[NCMS, pages 5, 6, 7] discusses “main programs” and “subprograms.” It is provided that the first word on the first line of a file containing a main program or a subprogram should be an O word. This word is to be considered to be the name of the file for calling it from another program. The last line of a file for a main program should have M2, M30, or M99 on it. The interpreter does not read O words, does not deal with subprograms, and does not recognize M99.

[EIA] does not discuss programs, so it has nothing to add to the above.

5.9 Control Panel Switches

The EMC controller is sensitive to all switches on the control panel. The interpreter needs to know the setting of only one switch, block delete. The other switches are handled by the EMC controller without the interpreter knowing what the settings are.

5.9.1 Block Delete Switch

If the block delete switch is on, the interpreter skips lines which start with a slash (the block delete character). Internally, the interpreter reads the line but does not try to figure out what it means. If the switch is off, lines starting with a slash are processed as though the slash was not there.

When the interpreter is used as part of the EMC control system, information about the block delete switch setting is passed to the interpreter during the initialization of executing a program. The interpreter does not check again, so changing the setting of that switch during the execution of a program will not change the way the program is executed.

When the interpreter is used stand-alone, a block delete switch setting may be included in the setup file. The default setting of the block delete switch is off, as shown in Table 3. The default is used when a setting is not specified in a setup file.

5.9.2 Other Switches

The speed or feed override switches on the control panel let the operator specify that the actual feed rate or spindle speed used in machining should be some percentage of the programmed rate. The EMC control system reacts to the setting of the speed or feed override switches on the control panel, when those switches are enabled. It does this, however, without the interpreter ever knowing their settings. As mentioned elsewhere, the interpreter will interpret M48 and M49 commands which enable or disable the switches, but it does not need to know what their settings are to do that. In Table 3 there is no representation of the settings of these switches.

The optional program stop switch on the machine console works as follows. If this switch is on **and** an input line contains an M1 code, program execution is supposed to stop until the cycle start button is pushed. The interpreter interprets an M1 on an input line into an OPTIONAL_PROGRAM_STOP canonical command in the output, as described elsewhere in this report. The interpreter itself has no knowledge of the setting of the optional program stop switch. In Table 3 there is no representation of the setting of this switch. It is up to the rest of the EMC control system to check the optional stop switch when the optional_program_stop canonical command is executed and either stop or not.

6 Capabilities of the RS274/VGER Interpreter

The interpreter implements only a portion of the RS274/NGC language. The current version includes the following capabilities. Any capability not explicitly included is excluded. Trying to use any excluded capability in an NC program will cause an error in the interpreter.

6.1 Words Recognized

The interpreter recognizes words beginning with the letters shown in Table 6. The meanings of the letters are as given in [NCMS, pages 8 - 11 and elsewhere]:

Letter	Meaning
A	A-axis of machine
C	C-axis of machine
D	tool radius compensation number
F	feedrate
G	general function (see below)
H	tool length offset
I	X-axis offset for arcs X offset in G87 canned cycle
J	Y-axis offset for arcs Y offset in G87 canned cycle
K	Z-axis offset for arcs Z offset in G87 canned cycle
L	number of repetitions in canned cycles key used with G10
M	miscellaneous function (see below)
N	line number
P	dwell time in canned cycles dwell time with G4 key used with G10
Q	feed increment in G83 canned cycle
R	arc radius canned cycle plane
S	spindle speed
T	tool selection
X	X-axis of machine
Y	Y-axis of machine
Z	Z-axis of machine

Table 6. Letters Recognized by the Interpreter

6.2 Input G Codes and M Codes

This section describes the specific G codes and M codes which are implemented in the interpreter.

6.2.1 G Codes Implemented

The G codes shown in Table 7 have been implemented. Unless noted otherwise in this document, implementation is as described in [NCMS]:

G Code	Meaning
G0	rapid positioning
G1	linear interpolation
G2	circular/helical interpolation (clockwise)
G3	circular/helical interpolation (counterclockwise)
G4	dwell
G10	coordinate system origin setting
G17	xy plane selection
G18	xz plane selection
G19	yz plane selection
G20	inch system selection
G21	millimeter system selection
G28	return to home
G30	return to secondary home
G38	straight probe
G40	cancel cutter diameter compensation
G41	start cutter diameter compensation left
G42	start cutter diameter compensation right
G43	tool length offset (plus)
G49	cancel tool length offset
G53	motion in machine coordinate system
G54	use preset work coordinate system 1
G55	use preset work coordinate system 2
G56	use preset work coordinate system 3
G57	use preset work coordinate system 4
G58	use preset work coordinate system 5
G59	use preset work coordinate system 6
G59.1	use preset work coordinate system 7
G59.2	use preset work coordinate system 8
G59.3	use preset work coordinate system 9
G80	cancel motion mode (including any canned cycle)
G81	drilling canned cycle
G82	drilling with dwell canned cycle
G83	chip-breaking drilling canned cycle
G84	right hand tapping canned cycle
G85	boring, no dwell, feed out canned cycle
G86	boring, spindle stop, rapid out canned cycle
G87	back boring canned cycle
G88	boring, spindle stop, manual out canned cycle
G89	boring, dwell, feed out canned cycle
G90	absolute distance mode
G91	incremental distance mode
G92	offset coordinate systems
G92.2	cancel offset coordinate systems
G93	inverse time feed mode
G94	feed per minute mode
G98	initial level return in canned cycles
G99	R-point level return in canned cycles

Table 7. G Codes Implemented in the Interpreter

6.2.2 Input M Codes Implemented

The following M codes are implemented as described in [NCMS], except that M30 also does a pallet shuttle and M60 is a language extension as described in Section 1.2.4.

M Code	Meaning
M0	program stop
M1	optional program stop
M2	program end
M3	turn spindle clockwise
M4	turn spindle counterclockwise
M5	stop spindle turning
M6	tool change
M7	mist coolant on
M8	flood coolant on
M9	mist and flood coolant off
M30	program end, pallet shuttle, and reset
M48	enable speed and feed overrides
M49	disable speed and feed overrides
M60	pallet shuttle and program stop

Table 8. M Codes Implemented in the Interpreter

7 Limitations of the Interpreter

The interpreter implements the parts of RS274/NGC which are expected to be most heavily used, but this includes only about half the language. This section calls out the major limitations of the interpreter.

Many G Codes and M Codes are not implemented. The interpreter handles 47 G codes; [NCMS] defines just over 100 G codes. The interpreter handles 13 of the 20 M codes defined in [NCMS], plus one not defined there. None of the canned cycles for milling are implemented, none of the skip functions are implemented, paramacros are not implemented, most of the setup commands are not implemented, and computational programming is not implemented.

No commands for transfer of control are implemented [NCMS, pages 74 - 76]. This includes: conditional operators and GOTO, IF-GOTO, and WHILE-DO-END commands.

The interpreter does not save input. This means that RS274/VGER programs cannot be written using the interpreter, which is of little importance. It also means that there is no record of what the user did, which is more important. The manual makes no provisions regarding saving input.

The manual [NCMS, pages 47, 49, 53 - 58] specifies the meaning of many parameters. While these parameters exist in the interpreter, only a modest fraction of them have the meaning specified in the manual, and are initialized or maintained. The manual specifies that some parameter values should be saved even when the power goes off [NCMS, page 54]. The interpreter saves no parameter values.

References

[Albus] Albus, James S; et al; *NIST Support to the Next Generation Controller Program: 1991 Final Technical Report*; NISTIR 4888; National Institute of Standards and Technology, Gaithersburg, MD; July 1992

[Allen Bradley] Allen Bradley; *RS274/NGC for the Low End Controller*; First Draft; Allen Bradley; August 1992

[EIA] Electronic Industries Association; *EIA Standard EIA-274-D Interchangeable Variable Block Data Format for Positioning, Contouring, and Contouring/ Positioning Numerically Controlled Machines*; Electronic Industries Association; Washington, DC; February 1979

[Fanuc] Fanuc Ltd.; *Fanuc System 9-Model A Operators Manual*; Pub B-52364E/03; Fanuc Ltd; 1981

[Kramer1] Kramer, Thomas R.; Proctor, Frederick M.; Michaloski, John L.; *The NIST RS274/NGC Interpreter, Version 1*; NISTIR 5416; National Institute of Standards and Technology, Gaithersburg, MD; April 1994

[Kramer2] Kramer, Thomas R.; Proctor, Frederick M.; *The NIST RS274KT Interpreter*; NISTIR 5738; National Institute of Standards and Technology, Gaithersburg, MD; October 1995

[Kramer3] Kramer, Thomas R.; Proctor, Frederick M.; *The NIST RS274/NGC Interpreter - Version 2*; NISTIR 5739; National Institute of Standards and Technology, Gaithersburg, MD; October 1995

[Proctor] Proctor, Frederick M.; Kramer, Thomas R.; Michaloski, John L.; *Canonical Functions for 4-Axis Machining*; NISTIR in draft; National Institute of Standards and Technology, Gaithersburg, MD; September 1995

[K&T] Kearney and Trecker Co.; *Part Programming and Operating Manual, KT/CNC Control, Type C*; Pub 687D; Kearney and Trecker Corp.; 1980

[Martin] Martin Marietta; *Controls Standardized Application (CSA)*; Draft Volume V of Next Generation Workstation/Machine Controller (NGC) Specification for an Open System Architecture Standard (SOSAS); Martin Marietta Document No. NGC-0001-14-000-CSA; March 1992

[Monarch] Monarch Cortland; *Programming Manual for Monarch VMC-75 with General Electric 2000MC Controls*; Monarch Cortland Publication Number PRG GE2000MC-4

[NCMS] National Center for Manufacturing Sciences; *The Next Generation Controller Part Programming Functional Specification (RS-274/NGC)*; Draft; NCMS; August 1994

[Trellis] Trellis Software and Controls, Inc.; *Trellis MAKEMESS Message Processing Utility User's Guide, Version 1.0*; Part Number TD-1012; Trellis Software and Controls, Inc.; 1995

Appendix A Software Details

This appendix describes the software for the interpreter. The appendix is intended for users and programmers who might want to modify the software or simply understand it.

A.1 Software Modules and Function Call Hierarchies

The interpreter is written in C++. Two methods of using the interpreter, stand-alone and integrated with the rest of EMC, are provided. The use of programs files differs between the two methods. Some code is common to both, and some code differs. Additionally, the stand-alone uses one set of files for error handling when compiled on a PC, but a different set when compiled on a Sun.

The program files are:

1. rs274vger.cc and its header file, rs274vger.hh — 10228 lines
2. canon.cc and its header file, canon.hh — 830 lines
3. driver.cc and its header file, nml_emc.hh — 1069 lines

4a. (for Sun) nce_err_sun.c and nce_code.h — 497 lines
4b. (for PC) nce_err.c and nce_code.h — 3890 lines

The organization of the software by module and file is shown in Figure 1. In the figure, arrows show the direction of function calls. The stand-alone interpreter uses the files shown in the left and middle columns. The integrated interpreter uses the files shown in the right and middle columns. The stand-alone files are arranged so that they mimic the arrangement of the integrated interpreter. The mimicry is so that running the stand-alone provides as good a test as possible of running integrated. If there were no integrated interpreter, the EMC emulation functions and data shown in the left column would not be needed.

The rs274vger.cc file has two parts, kernel functions (about 88% of the file) and interface functions. In the file itself the interface functions are given after the kernel functions. As shown in Figure 1, kernel functions are called only by interface functions, while interface functions are called by and call back to either EMC functions (if integrated) or driver functions (in the stand-alone). Both kernel functions and interface functions call canonical functions.

The driver.cc file also has two parts. In the file itself, the driver functions are given after the EMC emulation functions. The driver functions provide the user interface, and the EMC emulation functions provide an EMC-like environment for modeling data about the machine tool and controller settings.

Different versions of the canonical functions are used in the stand-alone and integrated interpreters. The canonical functions used with the stand-alone simply print themselves, while those used integrated send messages describing themselves. The same header file is used with both versions of the functions.

As described in Appendix B.1.6, the nce_code.h and the nce_err.c files are prepared automatically by MAKEMESS from a text-like error message definition file, nce_err.mdf, prepared by the programmer. The nce_err_sun.c file is extracted automatically by a small lex program from the nce_code.h file.

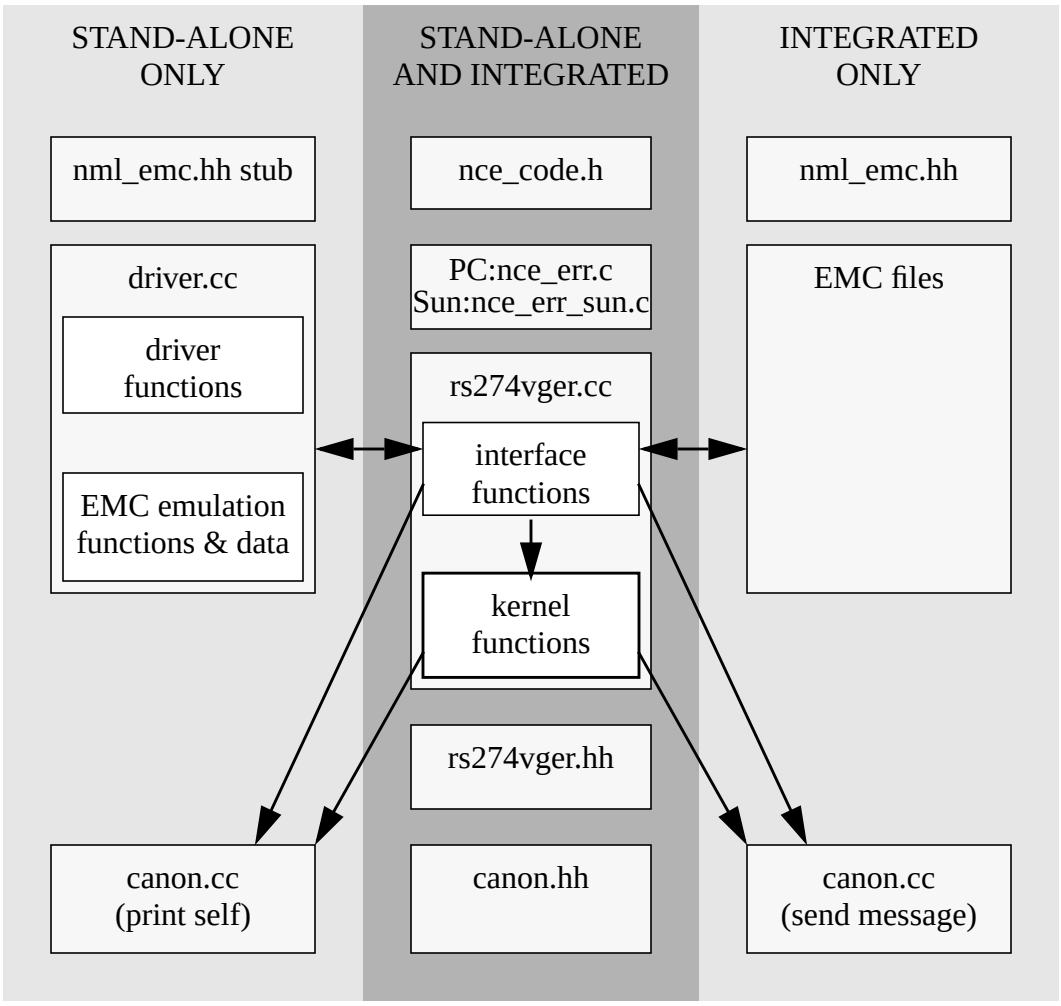

Figure 1. Interpreter Software

Function call hierarchies are shown in the following five figures. Figure 2, Figure 3, and Figure 4 show kernel function calls. All kernel functions appear on one or both of the two figures. Figure 5 shows the function calls from the interface functions. Figure 6 shows the hierarchy of function calls from the driver.

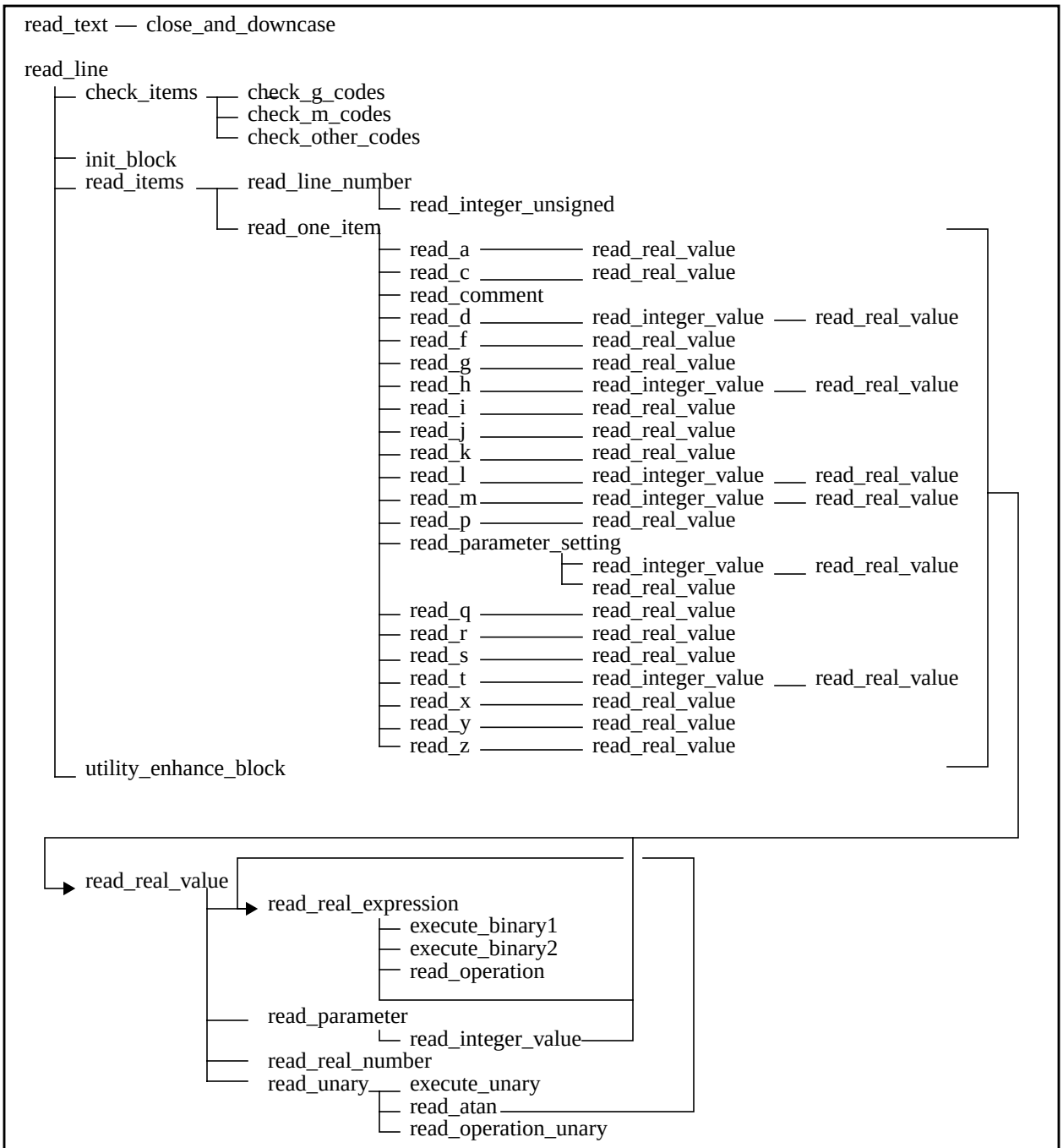

Figure 2. Interpreter Kernel Function Call Hierarchy (all but execute_block)

This shows the hierarchy of function calls from interpreter kernel functions read_text and read_line defined in rs274vger.cc to other kernel functions (excluding utility_error_number and nml_log_error - see text). Canonical functions calls are not shown. The hierarchy of calls from execute_block is shown in Figure 3 and Figure 4.

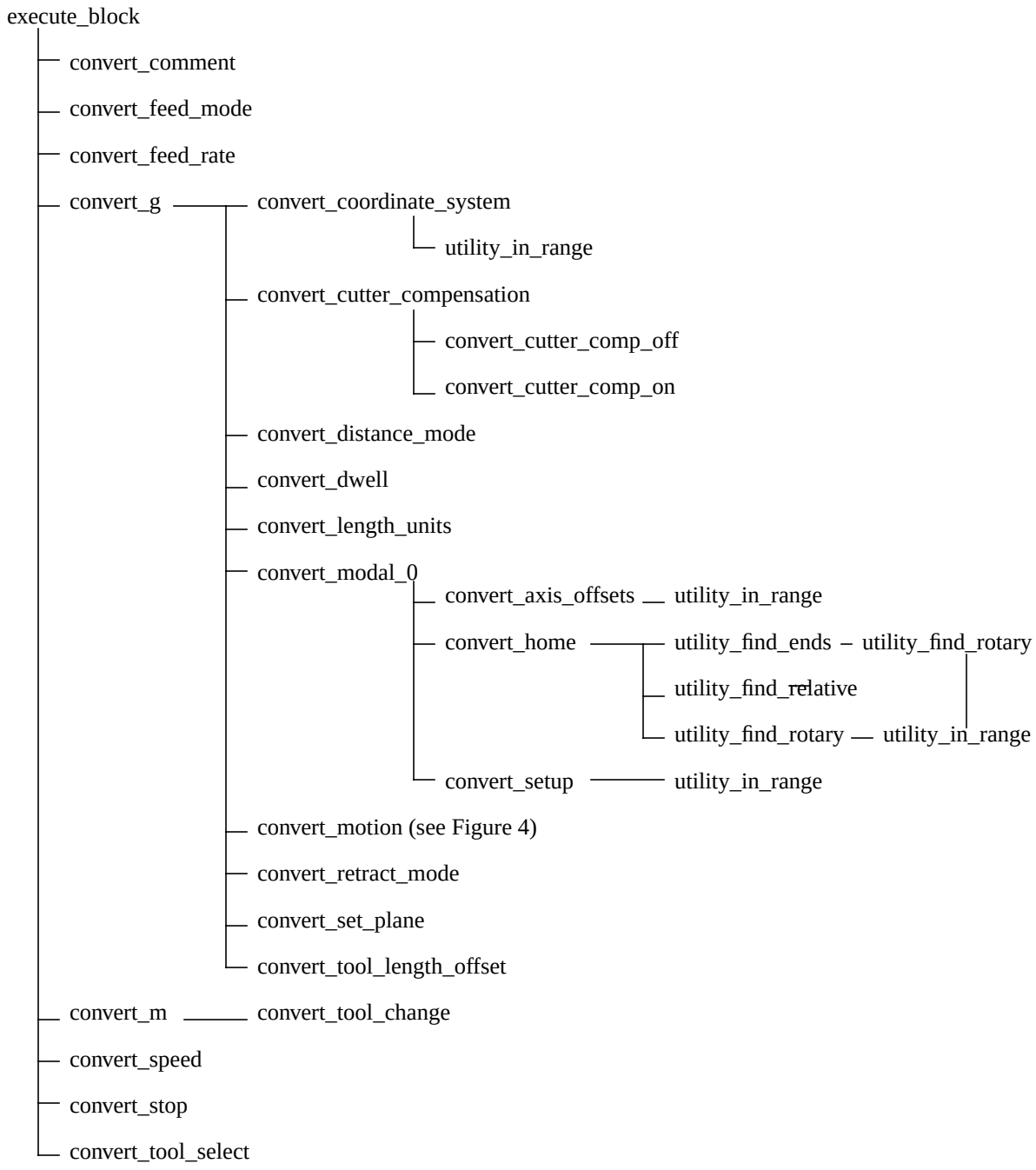

Figure 3. Interpreter Kernel Function Call Hierarchy (execute_block only)

This shows function calls from execute_block, defined in rs274vger.cc to other kernel functions (excluding utility_error_number and nml_log_error - see text). Canonical function calls are not shown. The hierarchy of calls from other top-level kernel functions is shown in Figure 2.

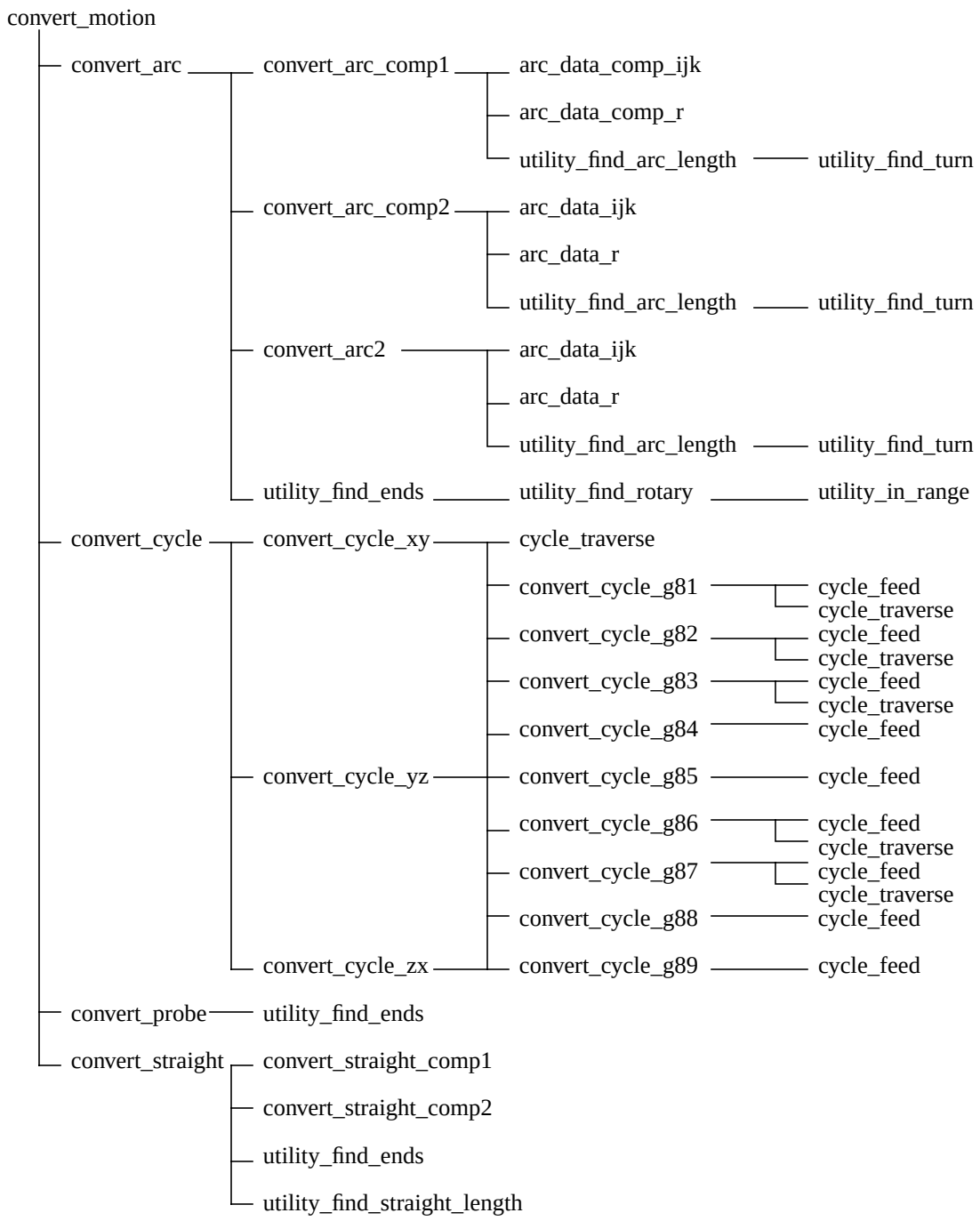

Figure 4. Interpreter Kernel Function Call Hierarchy (convert_motion only)

This shows function calls from convert_motion, defined in rs274vger.cc to other kernel functions (excluding utility_error_number and nml_log_error - see text). Canonical function calls are not shown. Convert_motion is shown called by convert_g in Figure 3. The hierarchies of calls from top-level kernel functions are shown in Figure 2 and Figure 3.

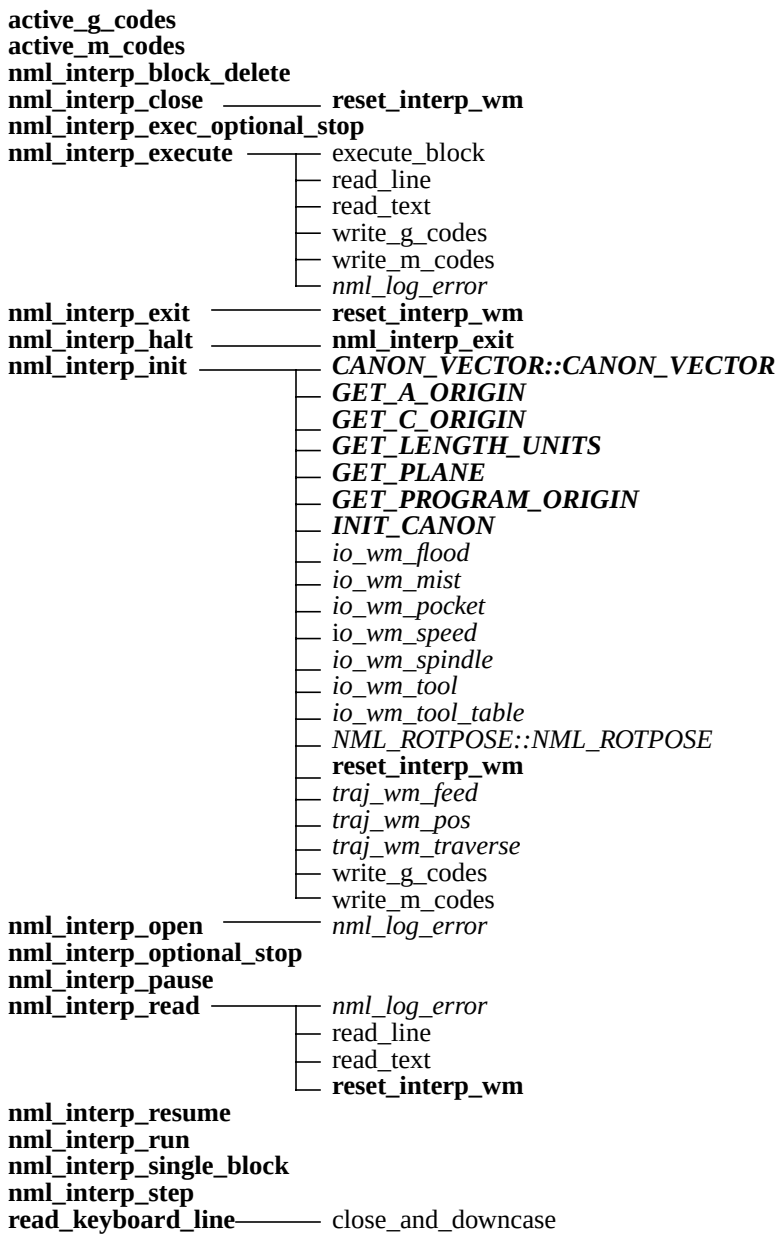

Figure 5. Interpreter Interface Function Call Hierarchy

This shows the hierarchy of function calls from interface functions in rs274vger.cc to:

1. kernel functions defined in rs274vger.cc (shown in ordinary typeface).

2. world modeling functions which are externally defined when the interpreter is integrated with the rest of EMC, but are defined in driver.cc in the stand-alone (shown in *italic*)

3. other interface functions defined in rs274vger.cc (shown in **boldface**)

4. special functions (not canonical functions) defined in canon.cc (or, for CANON_VECTOR::CANON_VECTOR, in canon.hh) (shown in ***bold italic***).

Eight interface functions defined in rs274vger.cc but not used in the stand-alone are not shown.

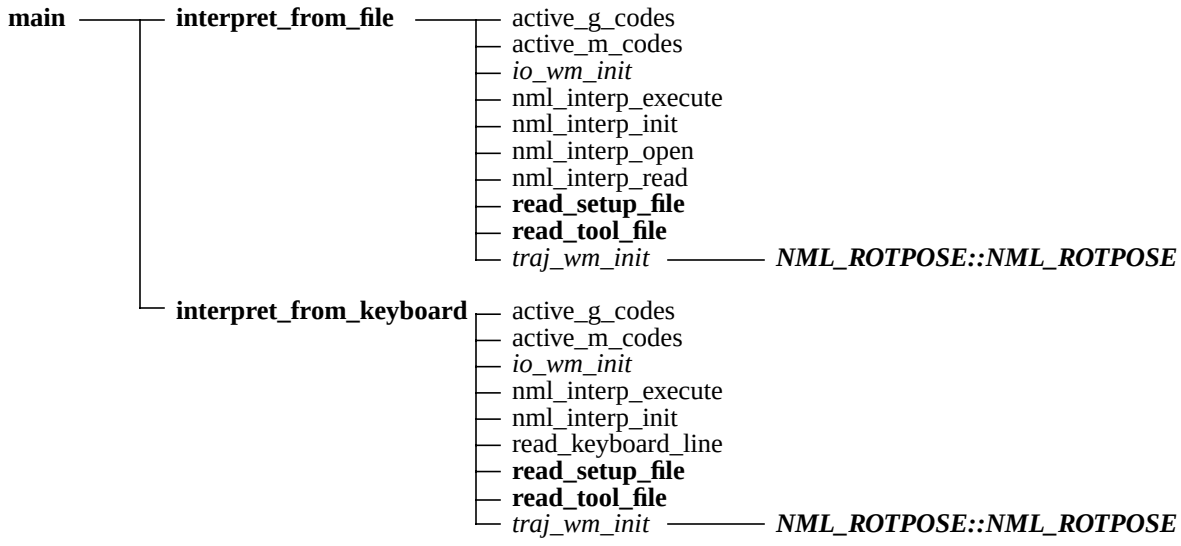

Figure 6. Interpreter Driver Function Call Hierarchy (for stand-alone interpreter)

This shows the hierarchy of function calls from main in the stand-alone driver file, driver.cc, to:

1. world modeling functions which are externally defined when the interpreter is integrated with the rest of EMC but are defined in driver.cc in the stand-alone (shown in *italic*)

2. other functions defined in driver.cc (shown in **boldface**)

3. interface functions defined in rs274vger.cc (shown in ordinary typeface).

A.2 Source Code Documentation

The source code is heavily documented. In general, for each function, four fields are given:

1. Returned Value - a description of possible returned values and the circumstances in which particular values may be returned. In most kernel functions, either RET_OK or some specific error code (which is a long unsigned int) may be returned.

2. Side Effects - a description of the important side effects (things other than the returned value) of executing a function. Since the returned value of most functions is used to indicate error status, the side effects of most functions are important.

3. Called By - a list of functions which call the function being documented.

4. Argument Values - a one-line description of the meaning of each argument to a function, placed immediately after the declaration of the argument. This field is omitted if there are no arguments.

In addition to these four fields, most functions have a paragraph or two (up to several pages) of discussion. Where a function implements an algorithm for geometric or numerical calculation (as do many of the functions having to do with cutter radius compensation, for example), the algorithm is described. Many citations to specific pages of the NCMS manual are included in these discussions.

Appendix B Functional Details

B.1 Error Handling and Exiting

The interpreter detects and flags most kinds of illegal input. Unreadable input, missing words, extra words, out-of-bounds numbers, and illegal combinations of words, for example, are all detected. The interpreter does not check for axis overtravel or excessively high feeds or speeds, however. The interpreter also does not detect situations where a legal command does something unfortunate, such as machining a fixture.

B.1.1 Basic Approach

The basic approach to error handling is:

1. Check carefully for errors.
2. If an error occurs, identify it specifically so that the user can be informed.
3. If an error occurs, return through the function call hierarchy rather than jumping out of it.

The data type used for error codes (and other return codes) differs between the kernel functions and the interface functions. Kernel functions return unsigned long ints - because the Trellis MAKEMESS utility is used, and it requires the use of unsigned long ints. Interface functions use ints. In the source code, unsigned long int is typedef'd as "how" (as in "how did it go?").

Error handling and exiting are handled together by having each function return a status value meaning one of three things:

1. No error has occurred and it is not time to exit (symbolic value of RET_OK in the kernel, OK in the interface).
2. It is time to exit (symbolic value of NCE_EXIT in the kernel, EXIT in the interface).
3. An error has occurred (symbolic value is a specific error code in the kernel, ERROR in the interface).

B.1.2 Error Messages

There are three sets of error messages:

1. input errors detected in the kernel or interface functions.
2. input errors detected in the stand-alone driver.
3. interpreter bugs detected in the kernel or interface functions (which should never occur).

The three sets of error messages are listed in the three subsections of Appendix E. Each message is intended to explain the error that triggered it clearly enough that a machine operator or NC programmer will be able to understand it and locate the error in the input. In the stand-alone interpreter, the line of NC code that caused the error will be printed, if the code is coming from a file. If the code is coming from the keyboard, it is already printed when the error is detected.

B.1.3 If an Error Occurs

If an error occurs, or if it is time to exit, control is passed back up through the function call hierarchy to some driver function (for the stand-alone) or to an EMC function (for the integrated interpreter). Section 2.1.1 describes interpreter behavior in case of an error. Kernel functions do not report errors directly (that is, they do not call an error reporting routine). Rather, if a kernel

function detects an error itself, the function stops where it is and returns the error code to the function that called it. If a subordinate kernel function called by a superior kernel function returns an error code, the superior stops where it is and passes the error code up to the function that called it. When the error code reaches an interface function, it is reported.

B.1.4 Handling Calculated Values

Since returned values are usually used as just described to handle the possibility of errors, an alternative method of passing calculated values is required. In general, if function A needs a value for variable V calculated by function B, this is handled by passing a pointer to V from A to B, and B calculates and sets V.

B.1.5 Compiler Macros

Handling errors detected in the kernel or interface functions (set 1 above) is done using compiler macros. Macros are used because a “return” statement may be included. As noted above, the function that initially detects an error simply returns an error code. Where one kernel function calls a second kernel function and the second function returns an error code, the first function uses a compiler macro called ERROR_MACRO_PASS. All that macro does is return the error code received from the second function.

If an interface function detects an error itself or is returned an error code from a kernel function, it calls ERROR_REPORT_MACRO. Two different versions of that macro are used, one for compiling on a PC (which includes calls to functions in the MAKEMESS library), and one for compiling on a Sun. Both versions call the nml_log_error function. Both ERROR_MACRO_PASS and ERROR_REPORT_MACRO have a function_name argument, which is not currently being used. Different versions of nml_log_error are used the stand-alone and integrated interpreters. In the stand-alone, nml_log_error prints the error to stderr.

Reporting errors which are interpreter bugs (set 3 above) is handled by the compiler macro BUG_MACRO. This behaves the same as ERROR_MACRO_PASS. However, all the messages which can be sent via the BUG_MACRO include the name of the function in which the bug occurred.

Reporting input errors in the driver is handled by the macro DRIVER_ERROR, which takes both a message string and a value as arguments, so that it can identify the error more precisely. It prints messages to stderr and returns ERROR.

B.1.6 Use of MAKEMESS

Error handling is managed with the help of the Trellis Software and Controls Company’s MAKEMESS facility [Trellis]. MAKEMESS has two parts, a pre-source code file processor, and a library of functions (which is available only in object code). To use MAKEMESS, the programmer writes a message definition file. In the message definition file (nce_err.mdf, not distributed with the source code for the interpreter), each error message is given a symbolic name. The programmer uses that name in source code to refer to the error message. Functions in the MAKEMESS library may be called in source code (for reporting an error, for example). The MAKEMESS processor reads the message definition file and writes two C language source files: a header file (nce_code.h) and a file of function definitions (nce_err.c).

Both the integrated interpreter and the stand-alone use the nce_code.h header file generated by MAKEMESS. Because the MAKEMESS library is available for PC's but not for Sun computers, and we are using Sun computers as well as PC's, the stand-alone interpreter does not use the library functions. Rather, an array of error messages (nce_err_sun.c) is generated automatically by a short lex program from the header file generated by MAKEMESS. The array index of each error message may be computed easily from the value of its symbolic name. Error reporting in the stand-alone interpreter is performed using this array. Error reporting in the integrated interpreter is performed using a MAKEMESS library function. The C source function definition file, nce_err.c, written by MAKEMESS is used in the integrated interpreter. Since the stand-alone does not use the MAKEMESS library, it can be compiled on a Sun, a PC, or any other computer for which a C++ compiler is available.

B.2 Cyclic Operation

In the integrated interpreter, input may be taken from a file or via manual data input (MDI) from the controller console. In the stand-alone, input may be taken from a file or from the keyboard of the computer running the interpreter. In all cases, two major phases of interpretation take place. The first phase consists of reading a line of code, checking it somewhat, and storing the information from the line in a structure called a "block". The second phase, which is always started by a call to nml_interp_execute, consists of examining the block, checking it further, and making calls to canonical functions. With MDI input the nml_interp_execute function triggers both phases. With file input, in both the stand-alone and the integrated version, the first phase is started by a call to nml_interp_read. In the stand-alone with keyboard input, the first phase is started by a call to read_keyboard_line.

B.2.1 Read, Store, and Check

The software reads lines of RS274/VGER code one at a time. First the line is read into a buffer. Next, any spaces not in comments are removed, and any upper case letters not in comments are changed to lower case letters.

Then the buffer and a counter holding the index of the next character to be read are handed around among a lot of functions named read_*XXXX*. All such functions read characters from the buffer using the counter. They all reset the counter to point at the character in the buffer following the last one used by the function. The first character read by most of these functions is expected to be a member of some set of characters (often a specific single character), and each function checks the first character.

Each line of code is stored until it has been executed in a reusable "block" structure, which has a slot for every potential piece of information on the line. Each time a useful piece of information has been extracted from the line, the information is put into the block.

The read_*XXXX* functions do a lot of error detection, but they only look for errors which would cause reading or storing to fail. After reading and storing is complete, more checking functions (check_g_codes, check_m_codes, and check_other_codes) are run. These functions look for logical errors, such as all axis words missing with a G code for motion in effect.

B.2.2 Execute

Once a block is built and checked, the block is executed by the execute_block function, which

calls one or more functions named convert_XXXX. In RS274/VGER, as in all dialects of RS274, a line of code may specify several different things to do, such as moving from one place to another along a straight line or arc, changing the feed rate, starting the spindle turning, etc. The order of execution of items in a block is critical to safe and effective machine operation (it is a good idea to start the spindle before cutting, for example), but is not specified clearly in the manual.

In the interpreter, items are executed in the order shown in Table 9 if they occur in the same block. Many of the kernel functions rely implicitly on this order being used. The source code documentation does not generally call out the fact that some other operation must have occurred before the one being documented.

1. comment (includes message).
2. set feed mode (G93, G94 — inverse time or per minute).
3. set feed rate (F).
4. set spindle speed (S).
5. select tool (T).
6. change tool (M6).
7. spindle on or off (M3, M4, M5).
8. coolant on or off (M7, M8, M9).
9. enable or disable overrides (M48, M49).
10. dwell (G4).
11. set active plane (G17, G18, G19).
12. set length units (G20, G21).
13. cutter radius compensation on or off (G40, G41, G42)
14. cutter length compensation on or off (G43, G49)
15. coordinate system selection (G54, G55, G56, G57, G58, G59, G59.1, G59.2, G59.3).
16. set distance mode (G90, G91).
17. set retract mode (G98, G99).
18. home (G28, G30), change coordinate system data (G10) or axis offsets (G92, G92.2).
19. perform motion (G0 to G3, G80 to G89).
20. stop (M0, M1, M2, M30, M60).

Table 9. Order of Execution

B.3 Tool Change

A CHANGE_TOOL canonical command call is made when an M6 code occurs on a line. Since this is a canonical command, it is not specialized to a particular machine. When the tool change is complete, the following conditions should prevail.

- The spindle should be stopped.
- The tool that was selected (by a T word on the same line or on any line after the previous tool change) should be in the spindle. The T number is an integer giving the id of the tool, not its changer slot.

- If the selected tool was not in the spindle before the tool change, the tool that was in the spindle (if there was one) should be in its changer slot.
- The coordinate axes should be stopped in the same absolute position they were in before the tool change (but the spindle may be re-oriented).
- No other changes should be made.

The tool change may include axis motion while it is in progress. For any machine, this motion must be predictable. It is up to the system programmers and operators to ensure that any such motion will not damage the machine or the workpiece.

If the tooling data is incorrect, so that the tool that was supposed to be selected is not actually in the expected slot, the tool change operation is not expected to be able to detect this, and no error will necessarily result. If the machine is able to detect this error, however, it should be detected and program execution should stop.

B.4 Milling Arcs

In RS274/VGER code [NCMS, pages 20, 21] a circular or helical arc is specified using either G2 (clockwise arc) or G3 (counterclockwise arc). The axis of the circle or helix must be parallel to the X, Y, or Z-axis of the machine coordinate system. The axis (or, equivalently, the plane perpendicular to the axis) is selected with G17 (Z-axis, XY-plane), G18 (Y-axis, ZX-plane), or G19 (X-axis, YZ-plane). If the arc is circular, it lies in the selected plane.

In the RS274/VGER language, two formats are allowed for specifying an arc. We will call these the center format and the radius format. The interpreter converts an arc in either format to a canonical ARC_FEED command, as described in Table 2.

If a line of RS274/VGER code makes an arc and includes a rotary axis move (A-axis or C-axis or both), the rotary axis turns at a constant rate so that it starts and finishes when the XYZ motion starts and finishes.

B.4.1 Radius Format Arc

In the radius format, the coordinates of the end point of the arc in the selected plane are specified along with the radius of the arc. A positive radius indicates that the arc turns through 180 degrees or less, while a negative radius indicates a turn of 180 degrees to 359.999 degrees. If the arc is helical, the value of the end point of the arc on the coordinate axis parallel to the axis of the helix is also specified.

Here is an example of a radius format RS274/VGER command to mill an arc:
G17 G2 x 10 y 15 r 20 z 5

That means to make a clockwise (as viewed from the positive Z-axis) circular or helical arc whose axis is parallel to the Z-axis, ending where X=10, Y=15, and Z=5, with a radius of 20. If the starting value of Z is 5, this is a circular arc; otherwise it is a helical arc.

B.4.2 Center Format Arc

In the center format, the coordinates of the end point of the arc in the selected plane are specified along with the offsets of the center of the arc from the current location.

Here is an example of a center format RS274/VGER command to mill an arc:
G17 G2 x 10 y 15 i 7 j 8 z 5

That means to make a clockwise (as viewed from the positive z-axis) circular or helical arc whose axis is parallel to the Z-axis, ending where X=10, Y=15, and Z=5, with its center offset in the X direction by 7 units from the current X location and offset in the Y direction by 8 units from the current Y location. If the starting value of Z is 5, this is a circular arc; otherwise it is a helical arc. The radius of the arc is not specified, but it may be found easily as the distance from the center of the circle to either the current point or the end point of the arc. The interpreter calculates both these distances and signals an error if they differ by more than 0.0002 inch (if inches are being used) or 0.002 millimeter (if millimeters are being used).

B.5 Coordinate Systems

The handling of coordinate systems by the interpreter involves the RS274/VGER commands for setting coordinate systems (G10 L2), selecting coordinate systems (G54, G55, G56, G57, G58, G59, G59.1, G59.2, and G59.3), offsetting all coordinate systems (G92 and G92.2), and operating in the absolute machine coordinate system temporarily (G53).

The canonical machining functions view of coordinate systems is:

1. There are two coordinate systems: absolute and program.
2. All coordinate values are given in terms of the program coordinate system.
3. The offsets of the program coordinate system may be reset.

The RS274/VGER view of coordinate systems, as given in section 3.2 of the manual [NCMS] is:

1. There are ten coordinate systems: absolute and 9 program. The program coordinate systems are numbered 1 to 9.
2. You can switch among the 9 but not to the absolute one. G54 selects coordinate system 1, G55 selects 2, and so on through G56, G57, G58, G59, G59.1, G59.2, and G59.3.
3. You can set the offsets of the 9 program coordinate systems using G10 L2 P*n* (*n* is the number of the coordinate system) with values for the axes in terms of the absolute coordinate system.
4. The first one of the 9 program coordinate systems is the default.
5. Data for coordinate systems is stored in parameters [NCMS, pages 59 - 60].
6. G53 means to interpret coordinate values in terms of the absolute coordinate system for the one block in which G53 appears.
7. You can offset the current coordinate system using G92. This offset will then apply to all nine program coordinate systems. This offset may be cancelled with G92.2.

The approach used in the interpreter mates the canonical and VGER views of coordinate systems as follows:

During initialization, the first VGER coordinate system is selected (origin_vger in the machine model is set to 1) with the offsets for that system all set to 0.0.

If a G code in the range G54 - G59.3 is encountered in an NC program, the data from the appropriate VGER coordinate system is copied into the origin offsets used by the interpreter, a SET_ORIGIN_OFFSETS function call is made, and the current position is reset.

If a G10 L2 P*n* is encountered, the convert_setup function is called to reset the offsets of program

coordinate system *n*. If coordinate system *n* is the one in use, a SET_ORIGIN_OFFSETS function call is made, and the current position is reset.

If a G53 is encountered, the axis values given in that block are used to calculate what the coordinates are of that point in the current coordinate system, and a STRAIGHT_TRAVERSE or STRAIGHT_FEED function call to that point using the calculated values is made. No offset values are changed.

If a G92 or G92.2 is encountered, that is handled by the convert_axis_offsets function. A G92 results in an axis offset for each axis being calculated and stored in the machine model, a SET_ORIGIN_OFFSETS function call is made, and the current position is reset. The axis offsets are applied with all nine coordinate systems. Axis offsets are initialized to zero. A G92.2 results in the axis offsets for all axes being reset to zero, a SET_ORIGIN_OFFSETS function call is made, and the current position is reset.

B.6 Tool Length Offsets

Tool length offsets are given as positive numbers in the tool table. Using a tool length offset is programmed using G43 H*n*, where *n* is the desired table index. It is expected that all entries in this table will be positive. Using no tool length offset is programmed using G49 [NCMS, page 74]. The H number is checked for being a non-negative integer when it is read. The interpreter behaves as follows.

1. If G43 H*n* is programmed, A USE_TOOL_LENGTH_OFFSET(length) function call is made (where length is the value of the tool length offset entry in the tool table whose index is *n*), tool_length_offset is reset in the machine settings model, and the value of current_z in the model is adjusted. Note that *n* does not have to be the same as the slot number of the tool currently in the spindle.

2. If G49 is programmed, USE_TOOL_LENGTH_OFFSET(0.0) is called, tool_length_offset is reset to 0.0 in the machine settings model, and the value of current_z in the model is adjusted.

B.7 Inverse Time Feed Rate

In the RS274/VGER language, two feed modes are recognized: units per minute and inverse time [NCMS, pages 35 - 37]. Programming G94 starts the units per minute mode. Programming G93 starts the inverse time mode.

In units per minute feed mode, an F word (no, not *that* F word; we mean *feedrate*) is interpreted to mean the tool tip should move at a certain number of inches per minute, millimeters per minute, or degrees per minute, depending upon what length units are being used and which axis or axes are moving.

In inverse time feed mode, an F word is interpreted to mean the move should be completed in [one divided by the F word] minutes. For example, if the F word is 2.0, the move should be completed in half a minute.

The interpreter handles the inverse time feed rate mode internally. The canonical functions have no command for putting a machine into this mode. If the interpreter is compiled with DEBUG #defined, when the interpreter converts a G93 or G94 command, it prints a comment saying it is going into inverse time feed mode (if G93) or into units per minute feed mode (if G94).

As specified in the manual, when the interpreter is in inverse time feed mode, an F word must appear on every line which has a motion. For each programmed motion (G1, G2, or G3), the interpreter calculates what the feed rate must be in units per minute to accomplish the move in the specified time. A SET_FEED_RATE canonical command (which always means units per minute) is output by the interpreter with this calculated rate. The calculation is: multiply the input F word by the path length of tool tip motion appropriate to the given kind of motion.

Being in inverse time feed mode does not affect G0 (rapid traverse) motions.

When in the inverse time feed mode, the interpreter reads but then ignores F words which are on lines that do not have G1, G2, or G3. This is not an error, but should probably be made one.

B.8 Canned Cycles

The canned cycles G81 through G89 have been implemented [NCMS, pages 98 - 100]. The manual is very sketchy regarding what these cycles are supposed to do. This section describes how each cycle has been implemented. Two examples are given with the description of G81 below.

All canned cycles are performed with respect to the currently selected plane. Any of the three planes (XY, YZ, ZX) may be selected. No rotary axis motion is allowed during canned cycles, inverse time feed rate is not allowed (but the interpreter is currently not detecting this error), and cutter radius compensation is not allowed. Throughout this section, most of the descriptions assume the XY-plane has been selected. The behavior is always analogous if the YZ or ZX-plane is selected.

All canned cycles use X, Y, R, and Z values in the NC code. These values are used to determine X, Y, R, and Z positions. The R (usually meaning retract) position is along the axis perpendicular to the currently selected plane (Z-axis for XY-plane, X-axis for YZ-plane, Y axis for ZX-plane). Some canned cycles use additional arguments.

In incremental distance mode: when the XY-plane is selected, X, Y, and R values are treated as increments to the current position and Z as an increment from the Z-axis position before the move involving Z takes place; when the YZ or ZX-plane is selected, treatment of the axis values is analogous. In absolute distance mode, the X, Y, R, and Z values are absolute positions in the current coordinate system, and it does not matter which plane is selected.

The repeat feature has been implemented, wherein the L word represents the number of repeats [NCMS, page 99]. We are not allowing L=0, contrary to the manual. If the repeat feature is used, it is normally used in incremental distance mode, so that the same sequence of motions is repeated in several equally spaced places along a straight line. We are allowing L > 1 in absolute distance mode to mean “do the same cycle in the same place several times”, as provided in the manual, although this seems abnormal. Omitting the L value is equivalent to specifying L=1.

When L>1 in incremental mode with the XY-plane selected, the X and Y positions are determined by adding the given X and Y values either to the current X and Y positions (on the first go-around) or to the X and Y positions at the end of the previous go-around (on the second and successive go-arounds). The R and Z positions do not change during the repeats.

The height of the retract move at the end of each repeat (called “clear Z” in the descriptions below) is determined by the setting of the retract_mode: either to the original Z position (if that is

above the R position and the retract_mode is G98, OLD_Z), or otherwise to the R position. This is a slight departure from [NCMS, page 98], which does not require checking that the original Z position is above the R position.

B.8.1 Preliminary Motion

At the very beginning of the execution of any of the canned cycles, with the XY-plane selected, if the current Z position is below the R position, the Z axis is traversed to the R position. This happens only once, regardless of the value of L.

In addition, for each repeat as specified by L, one or two moves are made before the rest of the cycle:

1. a straight traverse parallel to the XY-plane to the given XY-position
2. a straight traverse of the Z-axis only to the R position, if it is not already at the R position.

B.8.2 G81 Cycle

The G81 cycle is intended for drilling.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate to the Z position.
2. Retract the Z-axis at traverse rate to clear Z.

Example 1. Suppose the current position is (1, 2, 3) and the XY-plane has been selected, and the following line of NC code is interpreted.

G90 G81 G98 X4 Y5 Z1.5 R2.8

This calls for absolute distance mode (G90) and OLD_Z retract mode (G98) and calls for the G81 drilling cycle to be performed once. The X value and X position are 4. The Y value and Y position are 5. The Z value and Z position are 1.5. The R value and clear Z are 2.8. Old Z is 3. The following moves take place.

1. a traverse parallel to the XY plane to (4,5,3).
2. a traverse parallel to the Z-axis to (4,5,2.8)
3. a feed parallel to the Z-axis to (4,5,1.5)
4. a traverse parallel to the Z-axis to (4,5,3)

Example 2. Suppose the current position is (1, 2, 3) and the XY-plane has been selected, and the following line of NC code is interpreted.

G91 G81 G98 X4 Y5 Z-0.6 R1.8 L3

This calls for incremental distance mode (G91) and OLD_Z retract mode (G98) and calls for the G81 drilling cycle to be repeated three times. The X value is 4, the Y value is 5, the Z value is -0.6 and the R value is 1.8. The initial X position is 5 (=1+4), the initial Y position is 7 (=2+5), the clear Z position is 4.8 (=1.8+3), and the Z position is 4.2 (=4.8-0.6). Old Z is 3.

The first move is a traverse along the Z axis to (1,2,4.8), since old Z < clear Z.

The first repeat consists of 3 moves.

1. a traverse parallel to the XY-plane to (5,7,4.8)
2. a feed parallel to the Z-axis to (5,7, 4.2)

3. a traverse parallel to the Z-axis to (5,7,4.8)

The second repeat consists of 3 moves. The X position is reset to 9 (=5+4) and the Y position to 12 (=7+5).

1. a traverse parallel to the XY-plane to (9,12,4.8)
2. a feed parallel to the Z-axis to (9,12, 4.2)
3. a traverse parallel to the Z-axis to (9,12,4.8)

The third repeat consists of 3 moves. The X position is reset to 13 (=9+4) and the Y position to 17 (=12+5).

1. a traverse parallel to the XY-plane to (13,17,4.8)
2. a feed parallel to the Z-axis to (13,17, 4.2)
3. a traverse parallel to the Z-axis to (13,17,4.8)

B.8.3 G82 Cycle

The G82 cycle is intended for drilling.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate to the Z position.
2. Dwell for the given number of seconds.
3. Retract the Z-axis at traverse rate to clear Z.

B.8.4 G83 Cycle

The G83 cycle is intended for deep drilling or milling with chipbreaking. The dwell in this cycle causes any long stringers (which are common when drilling in aluminum) to be cut off. This cycle takes a Q value which represents a “delta” increment along the Z-axis.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate downward by delta or to the Z position, whichever is less deep.
2. Dwell for 0.25 second.
3. Repeat steps 1 and 2 until the Z position is reached.
4. Retract the Z-axis at traverse rate to clear Z.

B.8.5 G84 Cycle

The G84 cycle is intended for right-hand tapping.

0. Preliminary motion, as described above.
1. Start speed-feed synchronization.
2. Move the Z-axis only at the current feed rate to the Z position.
3. Stop the spindle.
4. Start the spindle counterclockwise.
5. Retract the Z-axis at the current feed rate to clear Z.
6. If speed-feed synch was not on before the cycle started, stop it.
7. Stop the spindle.
8. Start the spindle clockwise.

B.8.6 G85 Cycle

The G85 cycle is intended for boring or reaming.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate to the Z position.
2. Retract the Z-axis at the current feed rate to clear Z.

B.8.7 G86 Cycle

The G86 cycle is intended for boring. This cycle uses a P value, where P specifies the number of seconds to dwell.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate to the Z position.
2. Dwell for the given number of seconds.
3. Stop the spindle turning.
4. Retract the Z-axis at traverse rate to clear Z.
5. Restart the spindle in the direction it was going.

B.8.8 G87 Cycle

The G87 cycle is intended for back boring. The cycle which has been implemented is a modified version of [Monarch, page 5-24] since [NCMS, pages 98 - 100] gives no clue as to what the cycle is supposed to do. [K&T] does not have a back boring cycle.

The situation is that you have a through hole and you want to counterbore the bottom of hole. To do this you put an L-shaped tool in the spindle with a cutting surface on the UPPER side of its base. You stick it carefully through the hole when it is not spinning and is oriented so it fits through the hole, then you move it so the stem of the L is on the axis of the hole, start the spindle, and feed the tool upward to make the counterbore. Then you stop the tool, get it out of the hole, and restart it.

This cycle uses I and J values to indicate the position for inserting and removing the tool. I and J will always be increments from the X position and the Y position, regardless of the distance mode setting, as implied in [NCMS, page 98]. This cycle also uses a K value to specify the position along the Z-axis of the top of counterbore. The K value is an absolute Z-value in absolute distance mode, and an increment (from the Z position) in incremental distance mode.

0. Preliminary motion, as described above.
1. Move at traverse rate parallel to the XY-plane to the point indicated by I and J.
2. Stop the spindle in a specific orientation.
3. Move the Z-axis only at traverse rate downward to the Z position.
4. Move at traverse rate parallel to the XY-plane to the X,Y location.
5. Start the spindle in the direction it was going before.
6. Move the Z-axis only at the given feed rate upward to the position indicated by K.
7. Move the Z-axis only at the given feed rate back down to the Z position.
8. Stop the spindle in the same orientation as before.
9. Move at traverse rate parallel to the XY-plane to the point indicated by I and J.
10. Move the Z-axis only at traverse rate to the clear Z.
11. Move at traverse rate parallel to the XY-plane to the specified X,Y location.

12. Restart the spindle in the direction it was going before.

B.8.9 G88 Cycle

The G88 cycle is intended for boring. This cycle uses a P value, where P specifies the number of seconds to dwell.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate to the Z position.
2. Dwell for the given number of seconds.
3. Stop the spindle turning.
4. Stop the program so the operator can retract the spindle manually.
5. Restart the spindle in the direction it was going.

B.8.10 G89 Cycle

The G89 cycle is intended for boring. This cycle uses a P value, where P specifies the number of seconds to dwell.

0. Preliminary motion, as described above.
1. Move the Z-axis only at the current feed rate to the Z position.
2. Dwell for the given number of seconds.
3. Retract the Z-axis at the current feed rate to clear Z.

B.9 Probing

The RS274/VGER Interpreter implements a very simple form of probing, which is to call the canonical function STRAIGHT_PROBE when G38 is interpreted. The results of the probing operation do not affect the operation of the machining center (as would be the case if the results of probing were saved by resetting parameter values, for example). It is expected that the probe results are just collected and used in some way that does not affect the interpretation of the program that is running.

A more sophisticated form of probing has been implemented in version 3 of the four-axis RS274/NGC interpreter, in which the TURN_PROBE_ON and TURN_PROBE_OFF canonical commands are defined and used, the results of probing do affect machine operation by resetting parameter values, and the matter of coordinating interpretation and execution has been handled.

Appendix C Cutter Radius Compensation

C.1 Introduction

The cutter radius compensation[1]capabilities of the interpreter enable the programmer to specify that a cutter should travel to the right or left of an open or closed contour in the XY-plane composed of arcs of circles and straight line segments. The contour may be the outline of material not to be machined away, or it may be a tool path to be followed by an exactly sized tool. Figure 7 shows two examples of the path of a tool cutting using cutter radius compensation so that it leaves a triangle of material remaining. The figure gives corner coordinates for one shaded triangle.

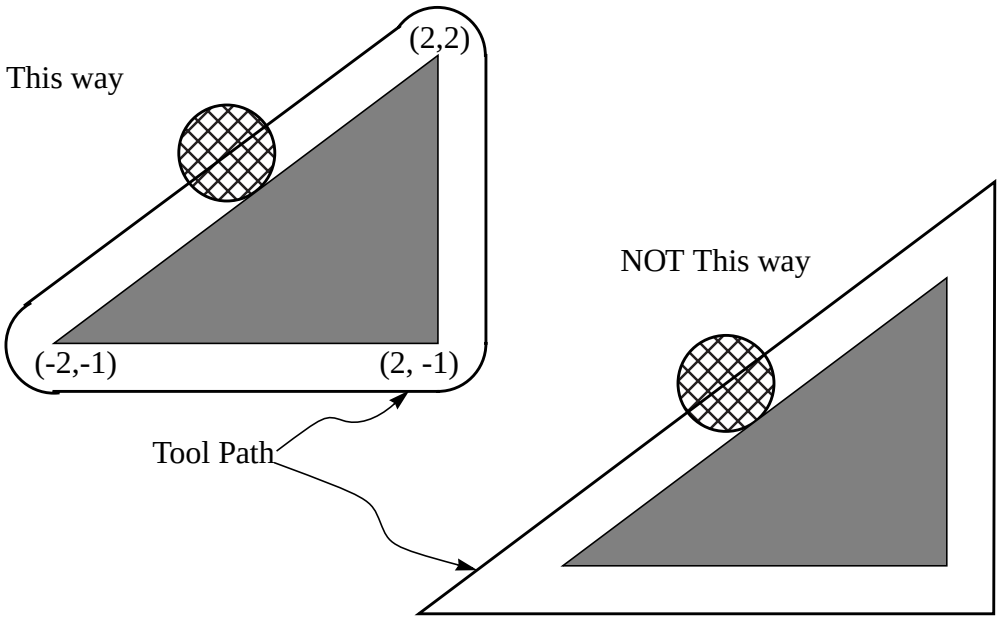

Figure 7. Two Cutter Radius Compensation Methods

In both examples, the shaded triangle represents material which should remain after cutting, and the line outside the shaded triangle represents the path of the tip of a cutting tool. Both paths will leave the shaded triangle uncut. The one on the left (with rounded corners) is the path the interpreter will generate. In the method on the right (the one not used), the tool does not stay in contact with the shaded triangle at sharp corners.

Z axis motion may take place while the contour is being followed in the XY plane. Portions of the contour may be skipped by retracting the Z axis above the part, following the contour to the next point at which machining should be done, and re-extending the Z-axis. These skip motions may be performed at feed rate (G1) or at traverse rate (G0).

Inverse time feed rate (G93) or units per minute feed rate (G94) may be used with cutter radius compensation. Under G94, the feed rate will apply to the actual path of the cutter tip, not to the

1. The term “cutter diameter compensation” is often used to mean the same thing.

programmed contour.

C.2 Programming Instructions

C.2.1 Turning Cutter Radius Compensation On

To start cutter radius compensation, program either G41 (for keeping the tool to the left of the contour) or G42 (for keeping the tool to the right of the contour). In Figure 7, for example, if G41 were programmed, the tool would stay left and move clockwise around the triangle, and if G42 were programmed, the tool would stay right and move counterclockwise around the triangle.

C.2.2 Turning Cutter Radius Compensation Off

To stop cutter radius compensation, program G40.

C.2.3 Sequencing

If G40, G41, or G42 is programmed in the same block as tool motion, cutter compensation will be turned on or off before the motion is made. To make the motion come first, the motion must be programmed in a separate, previous block.

C.2.4 Use of D Number

Programming the tool radius for cutter compensation may be done two ways: Either program a D number on the same line with G41 or G42, or program nothing.

If a D number is programmed, it must be a non-negative integer. It represents the slot number of the tool whose radius (half the diameter given in the tool table) will be used, or it may be zero (which is not a slot number). If it is zero, the value of the radius will also be zero. Any slot in the tool table may be selected this way. The D number does not have to be the same as the slot number of the tool in the spindle.

If a D number is not programmed, the slot number of the tool in the spindle will be used as the D number.

C.2.5 Tool Table

Cutter radius compensation uses data from the machining center’s tool table. For each slot in the tool carrousel, the tool table contains the diameter of the tool in that slot (or the difference between the actual diameter of the tool in the slot and its nominal value). The tool table is indexed by slot number. How to put data into the table when using the stand-alone interpreter is discussed in Appendix H.

C.3 Two Kinds of Contour

The interpreter handles compensation for two types of contour:

1. The contour given in the NC code is the edge of material that is not to be machined away. We will call this type a “material edge contour”.

2. The contour given in the NC code is the tool path that would be followed by a tool of exactly the correct radius. We will call this type a “tool path contour”.

The interpreter does not have any setting that determines which type of contour is used, but the description of the contour will differ (for the same part geometry) between the two types and the values for diameters in the tool table will be different for the two types.

C.3.1 Material Edge Contour

When the contour is the edge of the material, the outline of the edge is described in the NC program.

For a material edge contour, the value for the diameter in the tool table is the actual value of the diameter of the tool. The value in the table must be positive. The NC code for a material edge contour is the same regardless of the (actual or intended) diameter of the tool.

Example 1 :

Here is an NC program which cuts material away from the outside of the triangle in Figure 7. In this example, the cutter compensation radius is the actual radius of the tool in use, which is 0.5, The value for the diameter in the tool table is twice the radius, which is 1.0.

```
N0010 G41 G1 X2 Y2 (turn compensation on and make entry move)
N0020 Y-1 (follow right side of triangle)
N0030 X-2 (follow bottom side of triangle)
N0040 X2 Y2 (follow hypotenuse of triangle)
N0050 G40 (turn compensation off)
```

This will result in the tool following a path consisting of an entry move and the path shown on the left in Figure 7 going clockwise around the triangle. Notice that the coordinates of the triangle of material appear in the NC code. Notice also that the tool path includes three arcs which are not explicitly programmed; they are generated automatically.

C.3.2 Tool Path Contour

When the contour is a tool path contour, the path is described in the NC program. It is expected that (except for during the entry moves) the path is intended to create some part geometry. The path may be generated manually or by a post-processor, considering the part geometry which is intended to be made. For the interpreter to work, the tool path must be such that the tool stays in contact with the edge of the part geometry, as shown on the left side of Figure 7. If a path of the sort shown on the right of Figure 7 is used, in which the tool does not stay in contact with the part geometry all the time, the interpreter will not be able to compensate properly when undersized tools are used.

For a tool path contour, the value for the cutter diameter in the tool table will be a small positive number if the selected tool is slightly oversized and will be a small negative number if the tool is slightly undersized. As implemented, if a cutter diameter value is negative, the interpreter compensates on the other side of the contour from the one programmed and uses the absolute value of the given diameter. If the actual tool is the correct size, the value in the table should be zero.

Example 2 :

Suppose the diameter of the cutter currently in the spindle is 0.97, and the diameter assumed in generating the tool path was 1.0. Then the value in the tool table for the diameter for this tool

should be -0.03. Here is an NC program which cuts material away from the outside of the triangle in Figure 7:

```
N0010 G1 X1 Y4.5 (make alignment move)
N0020 G41 G1 Y3.5 (turn compensation on and make first entry move)
N0030 G3 X2 Y2.5 I1 (make second entry move)
N0040 G2 X2.5 Y2 J-0.5 (cut along arc at top of tool path)
N0050 G1 Y-1 (cut along right side of tool path)
N0060 G2 X2 Y-1.5 I-0.5 (cut along arc at bottom right of tool path)
N0070 G1 X-2 (cut along bottom side of tool path)
N0080 G2 X-2.3 Y-0.6 J0.5 (cut along arc at bottom left of tool path)
N0090 G1 X1.7 Y2.4 (cut along hypotenuse of tool path)
N0100 G2 X2 Y2.5 I0.3 J-0.4 (cut along arc at top of tool path)
N0110 G40 (turn compensation off)
```

This will result in the tool making an alignment move and two entry moves, and then following a path slightly inside the path shown on the left in Figure 7 going clockwise around the triangle. This path is to the right of the programmed path even though G41 was programmed, because the diameter value is negative.

C.4 Programming Errors and Limitations

The interpreter will issue the following messages involving cutter radius compensation.

1. Cannot change axis offsets with cutter radius comp
2. Cannot change units with cutter radius comp
3. Cannot turn cutter radius comp on out of XY-plane
4. Cannot turn cutter radius comp on when already on
5. Cannot use G28 or G30 with cutter radius comp
6. Cannot use G53 with cutter radius comp
7. Cannot use XZ plane with cutter radius comp
8. Cannot use YZ plane with cutter radius comp
9. Concave corner with cutter radius comp
10. Cutter gouging with cutter radius comp
11. D word on line with no cutter comp on (G41 or G42) command
12. Tool radius index too big
13. Tool radius not less than arc radius with cutter radius comp
14. Two G codes used from same modal group.

Most of these are self-explanatory. For those that require explanation, an explanation is given below.

Changing a tool while cutter radius compensation is on is not treated as an error, although it is unlikely this would be done intentionally. The radius used when cutter radius compensation was first turned on will continue to be used until compensation is turned off, even though a new tool is actually being used.

C.4.1 Concave Corner and Tool Radius Not Less than Arc Radius (9 and 13)

When cutter radius compensation is on, it must be physically possible for a circle whose radius is

the half the diameter given in the tool table to be tangent to the contour at all points of the contour. In particular, the interpreter treats concave corners and concave arcs into which the circle will not fit as errors, since the circle cannot be kept tangent to the contour in these situations. See Figure 8. This error detection does not limit the shapes which can be cut, but it does require that the programmer specify the actual shape to be cut (or path to be followed), not an approximation. In this respect, the RS274/VGER interpreter differs from interpreters used with many other controllers, which often allow these errors silently and either gouge the part or round the corner.

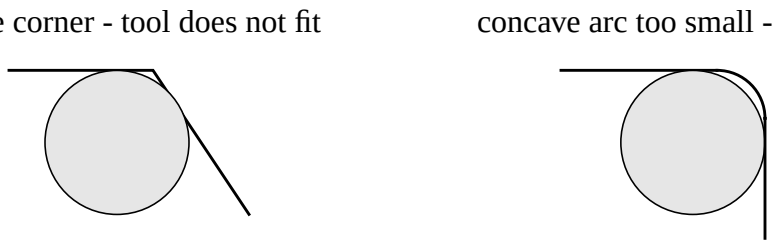

Figure 8. Two Cutter Radius Compensation Errors

In both examples, the line represents a contour, and the circle represents the cross section of a tool following the contour using cutter radius compensation (tangent to one side of the path.)

C.4.2 Cannot Turn Compensation on When Already On (4)

If cutter radius compensation has already been turned on, it cannot be turned on again. It must be turned off first; then it can be turned on again. It is not necessary to move the cutter between turning compensation off and back on, but the move after turning it back on will be treated as a first move, as described below.

It is not possible to change from one cutter radius index to another while compensation is on because of the combined effect of rules 4 and 11. It is also not possible to switch compensation from one side to another while compensation is on.

C.4.3 Cutter Gouging (10)

If the tool is already covering up the next XY destination point when cutter radius compensation is turned on, the gouging message is given when the line of NC code which gives the point is reached. In this situation, the tool is already cutting into material it should not cut. More details are given in Section C.5.

C.4.4 Tool Radius Index Too Big (12)

If a D word is programmed that is larger than the number of tool carrousel slots, this error message is given. In the current implementation, the number of slots is 68.

C.4.5 Two G Codes Used from Same Modal Group (14)

This is a generic message used for many sets of G codes. As applied to cutter radius compensation, it means that more than one of G40, G41, and G42 appears on a line of NC code. This is not allowed.

C.5 First Move into Cutter Compensation

C.5.1 Algorithm for First Move

The algorithm used for the first move when the first move is a straight line is to draw a straight line from the destination point which is tangent to a circle whose center is at the current point and whose radius is the radius of the tool. The destination point of the tool tip is then found as the center of a circle of the same radius tangent to the tangent line at the destination point. This is shown in Figure 9. If the programmed point is inside the initial cross section of the tool (the circle on the left), an error is signalled as described in Section C.4.3.

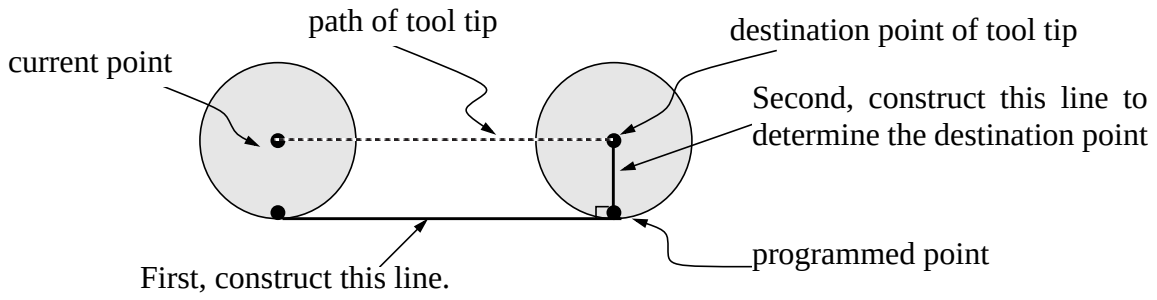

Figure 9. First Cutter Radius Compensation Move - Straight

If the first move after cutter radius compensation has been turned on is an arc, the arc which is generated is derived from an auxiliary arc which has its center at the programmed center point, passes through the programmed end point, and is tangent to the cutter at its current location. If the auxiliary arc cannot be constructed, an error is signalled. The generated arc moves the tool so that it stays tangent to the auxiliary arc throughout the move. This is shown in Figure 10.

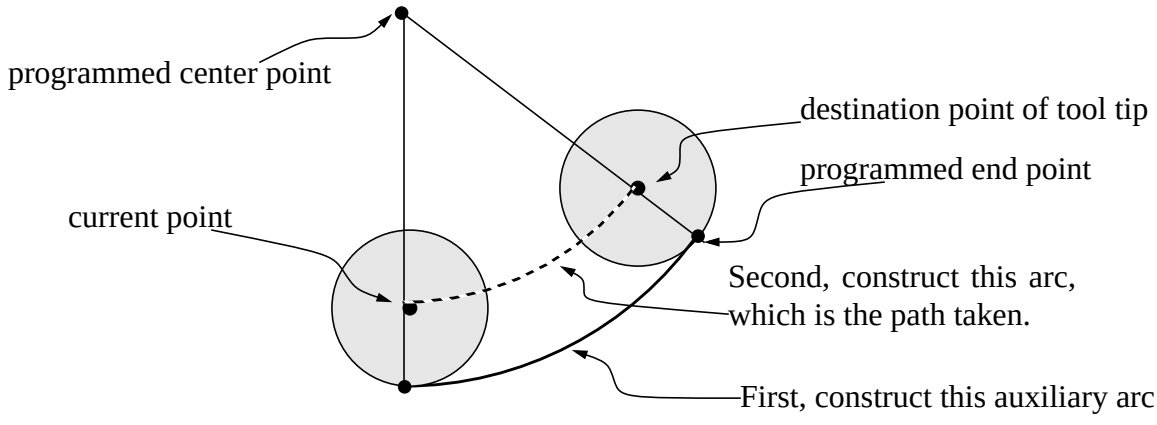

Figure 10. First Cutter Radius Compensation Move - Arc

Regardless of whether the first move is a straight line or an arc, the Z axis may also move at the same time. It will move linearly, as it does when cutter radius compensation is not being used.

Rotary axis motions (A and C axes) are allowed with cutter radius compensation, but using them would be very unusual.

After the entry moves of cutter radius compensation, the interpreter keeps the tool tangent to the programmed path on the appropriate side. If a convex corner is on the path, an arc is inserted to go around the corner. The radius of the arc is half the diameter given in the tool table.

When cutter radius compensation is turned off, no special exit move takes place. The next move is what it would have been if cutter radius compensation had never been turned on and the previous move had placed the tool at its current position.

If the interpreter has been compiled with the “DEBUG” option on, the interpreter signals when cutter radius compensation is turned on or off by calling the COMMENT canonical function with a message to that effect.

C.5.2 Programming Entry Moves

In general, an alignment move and two entry moves are needed to begin compensation correctly. However, where the programmed contour is a material edge contour and there is a convex corner on the contour, only one entry move (plus, possibly, a pre-entry move) is needed. The general method, which will work in all situations, is described first. We assume here that the programmer knows what the contour is already and has the job of adding entry moves.

C.5.2.1 General Method

The general method includes programming an alignment move and two entry moves.

The entry moves given in Section C.3.2 will be used as an example. Here is the relevant code again:

```
N0010 G1 X1 Y4.5 (make alignment move to point C)
N0020 G41 G1 Y3.5 (turn compensation on and make first entry move to point B)
N0030 G3 X2 Y2.5 I1 (make second entry move to point A)
```

See Figure 11. The figure shows the two entry moves but not the alignment move.

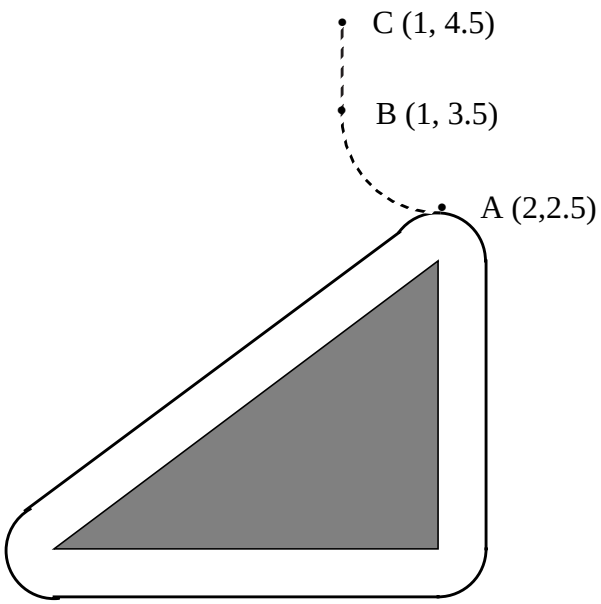

Figure 11. Cutter Radius Compensation Entry Moves

First, pick a point A on the contour where it is convenient to attach an entry arc. Specify an arc outside the contour which begins at a point B and ends at A tangent to the contour (and going in the same direction as it is planned to go around the contour). The radius of the arc should be larger than half the diameter given in the tool table. Then extend a line tangent to the arc from B to some point C, located so that the line BC is more than one radius long. After the construction is finished, the code is written in the reverse order from the construction. Cutter radius compensation is turned on after the alignment move and before the first entry move. In the code above, line N0010 is the alignment move, line N0020 turns compensation on and makes the first entry move, and line N0030 makes the second entry move.

In this example, the arc AB and the line BC are fairly large, but they need not be. For a tool path contour, the radius of arc AB need only be slightly larger than the maximum possible deviation of the radius of the tool from the exact size. Also for a tool path contour, the side chosen for compensation should be the one to use if the tool is oversized. As mentioned earlier, if the tool is undersized, the interpreter will switch sides.

C.5.2.2 Simple Method

If the contour is a material edge contour and there is a convex corner somewhere on the contour, a simpler method of making an entry is available. See Figure 12.

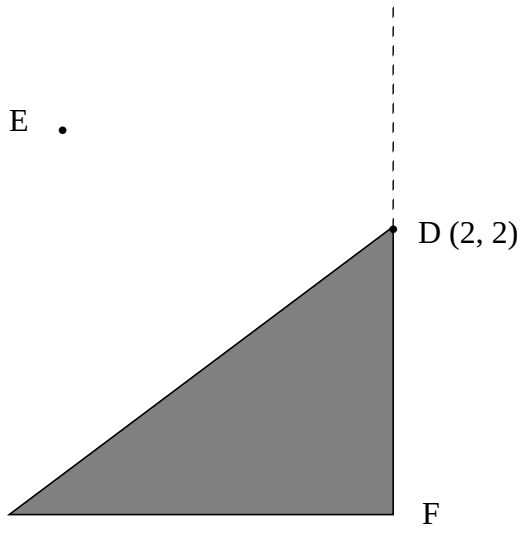

Figure 12. Simpler Cutter Radius Compensation Entry Move

First, pick a convex corner, D. Decide which way you want to go along the contour from D. In our example we are keeping the tool to the left of the contour and going next towards F. Extend the line FD (if the next part of the contour is an arc, extend the tangent to arc FD from D) to divide the area outside the contour near D into two regions. Make sure the center of the tool is currently in the region on the same side of the extended line as the material inside the contour near D. If not, move the tool into that region. In the example, point E represents the current location of the center of the tool. Since it is on the same side of line DF as the shaded triangle, no additional move is needed. Now write a line of NC code that turns compensation on and moves to point D.

`N0010 G41 G1 X2 Y2` (turn compensation on and make entry move)

This method will also work at a concave corner on a tool path contour, if the actual tool is oversized, but it will fail with a tool path contour if the tool is undersized.

C.6 Other Items

C.6.1 Where Cutter Radius Compensation is Performed

The complete set of canonical functions includes functions which turn cutter radius on and off, so that cutter radius compensation can be performed in the controller executing the canonical functions. In the RS274/VGER interpreter, however, these commands are not used. Compensation is done by the interpreter and reflected in the output commands, which continue to direct the

motion of the center of the cutter tip. This simplifies the job of the motion controller while making the job of the interpreter a little harder.

C.6.2 Algorithms for Cutter Radius Compensation

The algorithms for the linear first and last moves of cutter radius compensation used in the interpreter may differ from those in the manual [NCMS, pages 78 to 81], but these paths are not clearly described in the manual. The interpreter allows the entry and exit moves to be arcs, but the manual does not. The behavior for the intermediate moves is the same, except that some situations treated as errors in the interpreter are not treated as errors in the manual.

C.6.3 Data for Cutter Radius Compensation

The interpreter machine model keeps three data items for cutter radius compensation: the setting itself (right, left, or off), program_x, and program_y. The last two represent the X and Y positions which are given in the NC code while compensation is on. When compensation is off, these both are set to a very small number (10^{-20}) whose symbolic value (in a #define) is “unknown”. The interpreter machine model uses the data items current_x and current_y to represent the position of the center of the tool tip (in the currently active coordinate system) at all times.

Appendix D Transcript of a Session

This is a transcript of a session using the stand-alone interpreter with keyboard input. Characters entered by the user are shown in **boldface**. All user input is followed by a carriage return not shown here.

```
1} rs274vger
 1 N0 SET_FEED_REFERENCE(CANON_XYZ)
 2 N0 SET_ORIGIN_OFFSETS(0.0000, 0.0000, 0.0000, 0.0000, 0.0000)
name of tool file => ../tool/cds
name of setup file =>
using default machine setup
READ => g1 x3 y1 f20.0
EXEC <-;
 3 N ... SET_FEED_RATE(20.0000)
 4 N ... STRAIGHT_FEED(3.0000, 1.0000, 0.0000, 0, 0.0000, 0, 0.0000)
READ => g2 x[6-[4*3/2]] r 7.01 z0.5
EXEC <-;
 5 N ... ARC_FEED(0.0000, 1.0000, 1.5000, 7.8476, -1, 0.5000, 0,
      0.0000, 0, 0.0000)
READ => (that was a helical arc)
EXEC <-;
 6 N ... COMMENT("that was a helical arc")
READ => t2
EXEC <-;
 7 N ... SELECT_TOOL(2)
READ => m6 g43 h2
EXEC <-;
 8 N ... CHANGE_TOOL(2)
 9 N ... USE_TOOL_LENGTH_OFFSET(1.0000)
READ => m2
EXEC <-;
 10 N ... PROGRAM_END()
READ => g1 x asim[0.5]
EXEC <-;
interpreter error 136: Unknown word starting with A
READ => g1 x asin[0.5]
EXEC <-;
 11 N ... STRAIGHT_FEED(30.0000, 1.0000, -0.5000, 0, 0.0000, 0, 0.0000)
READ => g91 g81 x3 y2 z-0.8 r1.5 l2
EXEC <-;
 12 N ... COMMENT("interpreter: distance mode changed to incremental")
 13 N ... STRAIGHT_TRAVERSE(30.0000, 1.0000, 1.0000, 0, 0.0000, 0, 0.0000)
 14 N ... STRAIGHT_TRAVERSE(33.0000, 3.0000, 1.0000, 0, 0.0000, 0, 0.0000)
 15 N ... STRAIGHT_FEED(33.0000, 3.0000, 0.2000, 0, 0.0000, 0, 0.0000)
 16 N ... STRAIGHT_TRAVERSE(33.0000, 3.0000, 1.0000, 0, 0.0000, 0, 0.0000)
 17 N ... STRAIGHT_TRAVERSE(36.0000, 5.0000, 1.0000, 0, 0.0000, 0, 0.0000)
 18 N ... STRAIGHT_FEED(36.0000, 5.0000, 0.2000, 0, 0.0000, 0, 0.0000)
 19 N ... STRAIGHT_TRAVERSE(36.0000, 5.0000, 1.0000, 0, 0.0000, 0, 0.0000)
READ => quit
```

Appendix E Error Messages

The error messages used throughout the interpreter are intended to be self-explanatory. There are three categories of error message: interpreter kernel and interface input error messages, interpreter kernel internal error messages, and interpreter driver input error messages. These are described in Appendix E.1, Appendix E.2, and Appendix E.3, respectively. Each list is arranged alphabetically. Messages are in **boldface** type. Following each message in Appendix E.1 and Appendix E.3 is the name of the function or functions in which it is found, printed in *italics*.

E.1 Interpreter Kernel and Interface Input Error Messages

This is a list of all 154 input error messages in the interpreter kernel and interface. Each message describes some kind of error in the input to the interpreter. The number before each message is the index number of the message in the array of error messages in the nce_err_sun.c file. Each message has two other numbers of interest not given here: (1) the number given in the message definition file for MAKEMESS, nce_err.mdf and (2) the symbolic value for the error given in nce_code.h.

0. All axis values missing with G28 . . . . . *check_g_codes*
1. All axis values missing with G30 . . . . . *check_g_codes*
2. All axis values missing with G92 . . . . . *check_g_codes*
3. Arc radius too small to reach end point . . . . . *arc_data_r*
4. Argument to acos out of range. . . . . *execute_unary*
5. Argument to asin out of range . . . . . *execute_unary*
6. Attempt to divide by zero . . . . . *execute_binary1*
7. Bad character used . . . . . *read_one_item*
8. Bad format unsigned integer . . . . . *read_integer_unsigned*
9. Bad number format . . . . . *read_real_number*
10. Cannot change axis offsets with cutter radius comp . . . . . *convert_axis_offsets*
11. Cannot change units with cutter radius comp . . . . . *convert_length_units*
12. Cannot do G1 with zero feed rate . . . . . *convert_straight*
13. Cannot do zero repeats of cycle . . . . . *convert_cycle*
14. Cannot make arc with zero feed rate . . . . . *convert_arc*
15. Cannot put a C-axis command in a canned cycle . . . . . *check_other_codes*
16. Cannot put an A-axis command in a canned cycle . . . . . *check_other_codes*
17. Cannot raise negative number to non-integer power . . . . . *execute_binary1*
18. Cannot turn cutter radius comp on out of XY-plane. . . . *convert_cutter_compensation_on*
19. Cannot turn cutter radius comp on when already on . . . *convert_cutter_compensation_on*
20. Cannot use G28 or G30 with cutter radius comp . . . . . *convert_home*
21. Cannot use G53 in incremental distance mode . . . . . *check_g_codes*
22. Cannot use G53 with cutter radius comp . . . . . *convert_straight*
23. Cannot use XZ plane with cutter radius comp . . . . . *convert_set_plane*
24. Cannot use YZ plane with cutter radius comp . . . . . *convert_set_plane*
25. Cannot use a G code for motion with G10 . . . . . *check_g_codes*
26. Cannot use a G code for motion with G28 . . . . . *check_g_codes*
27. Cannot use a G code for motion with G30 . . . . . *check_g_codes*
28. Cannot use a G code for motion with G92 . . . . . *check_g_codes*

E.2 Interpreter Kernel Internal Error Messages

The following error messages should never appear but are provided as a check on the internal workings of the interpreter kernel. The name of the function in which the error has occurred is part of each message. The appearance of one of these means there is a bug in the source code for the interpreter. If one of these error messages ever appears, please contact kramer@cme.nist.gov by email; include the message and describe the circumstances in which it appeared.

The number before each message is the index number of the message in the array of error messages in the nce_err_sun.c file. Each message has two other numbers of interest not given here: (1) the number given in the message definition file for MAKEMESS, nce_err.mdf and (2) the symbolic value for the error given in nce_code.h..

200. Bad G code in modal group 0 in check_g_codes
201. Code is not G0 or G1 in convert_straight
202. Code is not G0 or G1 in convert_straight_comp1
203. Code is not G0 or G1 in convert_straight_comp2
204. Code is not G0 to G3 or G80 to G89 in convert_motion
205. Code is not G10, G28, G30, G92, G92.2 in convert_modal_0
206. Code is not G17, G18, or G19 in convert_set_plane
207. Code is not G2 or G3 in arc_data_comp_ijk
208. Code is not G2 or G3 in arc_data_ijk
209. Code is not G20 or G21 in convert_length_units
210. Code is not G28 or G30 in convert_home
211. Code is not G40, G41, or G42 in convert_cutter_compensation
212. Code is not G43 or G49 in convert_tool_length_offset
213. Code is not G54 to G59.3 in convert_coordinate_system
214. Code is not G90 or G91 in convert_distance_mode
215. Code is not G92 or G92.2 in convert_axis_offsets
216. Code is not G93 or G94 in convert_feed_mode
217. Code is not G98 or G99 in convert_retract_mode
218. Code is not M0, M1, M2, M30 or M60 in convert_stop
219. Convert_cycle_xy should not have been called
220. Convert_cycle_yz should not have been called
221. Convert_cycle_zx should not have been called
222. Distance mode is not absolute or incremental in convert_cycle_xy
223. Distance mode is not absolute or incremental in convert_cycle_yz
224. Distance mode is not absolute or incremental in convert_cycle_zx
225. Plane is not XY, YZ, or XZ in convert_arc
226. Plane is not XY, YZ, or XZ in convert_cycle
227. Read_a should not have been called
228. Read_c should not have been called
229. Read_comment should not have been called
230. Read_d should not have been called
231. Read_f should not have been called
232. Read_g should not have been called
233. Read_h should not have been called
234. Read_i should not have been called

235. Read_j should not have been called
236. Read_k should not have been called
237. Read_l should not have been called
238. Read_line_number should not have been called
239. Read_m should not have been called
240. Read_p should not have been called
241. Read_parameter should not have been called
242. Read_parameter_setting should not have been called
243. Read_q should not have been called
244. Read_r should not have been called
245. Read_real_expression should not have been called
246. Read_s should not have been called
247. Read_t should not have been called
248. Read_x should not have been called
249. Read_y should not have been called
250. Read_z should not have been called
251. Side fails to be right or left in convert_straight_comp1
252. Side fails to be right or left in convert_straight_comp2
253. Sscanf failure in read_integer_unsigned
254. Sscanf failure in read_real_number
255. Unknown operation in execute_binary1
256. Unknown operation in execute_binary2
257. Unknown operation in execute_unary

E.3 Interpreter Driver Input Error Messages

The following error messages may be printed by the stand-alone interpreter but not the integrated interpreter. Each message describes an error in the input to the stand-alone interpreter that is not an interpreter kernel error. The messages originate in the source code for the stand-alone interpreter driver. A word in ***bold italics*** stands for a number or word that will differ according the exact nature of the error. The error messages are not kept in an array.

1. Bad input line "*text_line*" in setup file . . . . . *read_setup_file*
2. Bad input line "*text_line*" in tool file. . . . . *read_tool_file*
3. Bad setup file format. . . . . *read_setup_file*
4. Bad tool file format . . . . . *read_tool_file*
5. Bad value *given_value* for block_delete in setup file . . . . . *read_setup_file*
6. Bad value *given_value* for coordinate system in setup file . . . . . *read_setup_file*
7. Bad value *given_value* for cutter_radius_comp in setup file. . . . . *read_setup_file*
8. Bad value *given_value* for distance_mode in setup file . . . . . *read_setup_file*
9. Bad value *given_value* for feed_mode in setup file. . . . . *read_setup_file*
10. Bad value *given_value* for flood in setup file. . . . . *read_setup_file*
11. Bad value *given_value* for length_units in setup file . . . . . *read_setup_file*
12. Bad value *given_value* for mist in setup file . . . . . *read_setup_file*
13. Bad value *given_value* for plane in setup file . . . . . *read_setup_file*
14. Bad value *given_value* for speed_feed_mode in setup file. . . . . *read_setup_file*
15. Bad value *given_value* for spindle_turning in setup file . . . . . *read_setup_file*
16. Cannot open *file_name* . . . . . *read_setup_file, read_tool_file*
17. Unknown attribute *attribute_name* in setup file. . . . . *read_setup_file*
18. Usage "rs274vger" or "rs274vger filename" or "rs274vger filename continue" *. . . main*

Appendix F Production Rules for Line Grammar and Syntax

The following is a production rule definition of what this RS274/VGER interpreter recognizes as valid combinations of symbols which form a readable line (the term “line” is at the top of this production hierarchy). The productions are arranged alphabetically to make connections easy to trace.

The productions are intended to be unambiguous. That is, no permissible line of code can be interpreted more than one way. To make the productions below entirely unambiguous, it is implicit that if a string of characters that can meet the requirements of an ordinary comment also can be interpreted as a message, that string is a message and not an ordinary comment.

The term comment_character is used in the productions but not defined there. A comment character is any printable character plus space and tab, except for a left parenthesis or right parenthesis. This implies comments cannot be nested.

It is implicit in the production rules that, except inside parentheses, space and tab characters may be ignored, or have been removed by pre-processing.

These production rules do not include constraints implied by the semantics of the interpreter. Most of the constraints are in terms of combinations of words (as defined below). Many lines of code that are readable under these production rules will not be executable because they violate constraints. Any constraint violation will be detected by the interpreter and will result in an error message. The error messages are included in Appendix E.

In the productions, a parameter_index is defined as a synonym for real_value. The constraint on a parameter_index is that it must be an integer between 1 and 5399 (inclusive).

F.1 Production Language

The symbols in the productions are mostly standard syntax notation. Meanings of the symbols follow.

- = The symbol on the left of the equal sign is equivalent to the expression on the right
- + followed by
- | or
- . end of production (a production may have several lines)
- [ ] zero or one of the expression inside square brackets may occur
- { } zero to many of the expression inside curly braces may occur
- ( ) exactly one of the expression inside parentheses must occur

F.2 Productions

Any term in this subsection that is used on the right of an equal sign but is not defined in this subsection (i.e., does not appear on the left in any definition) is defined in the next subsection in terms of characters.

arc_tangent_combo = arc_tangent + expression + divided_by + expression .
binary_operation1 = divided_by | modulo | power | times .

```
binary_operation2 = and | exclusive_or | minus |  non_exclusive_or | plus .
combo1 = real_value + { binary_operation1 + real_value } .
comment = message | ordinary_comment .
comment_character = see explanation above .
digit = zero | one | two | three | four | five | six | seven | eight | nine .
expression = left_bracket + (combo1 + { binary_operation2 combo1 }) + right_bracket .
letter_a = big_a | little_a .
letter_c = big_c | little_c .
letter_d = big_d | little_d .
letter_f = big_f | little_f .
letter_g = big_g | little_g .
letter_h = big_h | little_h .
letter_i = big_i | little_i .
letter_j = big_j | little_j .
letter_k = big_k | little_k .
letter_l = big_l | little_l .
letter_m = big_m | little_m .
letter_n = big_n | little_n .
letter_p = big_p | little_p .
letter_q = big_q | little_q .
letter_r = big_r | little_r .
letter_s = big_s | little_s .
letter_t = big_t | little_t .
letter_x = big_x | little_x .
letter_y = big_y | little_y .
letter_z = big_z | little_z .
line = [block_delete] + [line_number] + {segment} + end_of_line .
line_number = letter_n + digit + [digit] + [digit] + [digit] + [digit] .
message = left_parenthesis + {white_space} + letter_m + {white_space} + letter_s +
          {white_space} + letter_g + {white_space} + comma + {comment_character} +
          right_parenthesis .
mid_line_letter = letter_a | letter_c| letter_d | letter_f | letter_g | letter_h | letter_i | letter_j
          | letter_k | letter_l | letter_m | letter_p | letter_q | letter_r | letter_s | letter_t
          | letter_x | letter_y | letter_z .
mid_line_word = mid_line_letter + real_value .
ordinary_comment = left_parenthesis + {comment_character} + right_parenthesis .
ordinary_unary_combo =  ordinary_unary_operation + expression .
ordinary_unary_operation =
          absolute_value | arc_cosine | arc_sine | cosine | e_raised_to |
          fix_down | fix_up | natural_log_of | round | sine | square_root | tangent .
parameter_index = real_value .
parameter_setting = parameter_sign + parameter_index + equal_sign + real_value .
parameter_value = parameter_sign  + parameter_index .
real_number =
          [ plus | minus ] +
          (( digit + { digit } + [decimal_point] + {digit}) | ( decimal_point + digit + {digit})) .
```

real_value = real_number | expression | parameter_value | unary_combo .
segment = mid_line_word | comment | parameter_setting .
unary_combo = ordinary_unary_combo | arc_tangent_combo .
white_space = space | tab .

F.3 Production Tokens in Terms of Characters

We have omitted the letters and digits in the list below, since they are all the obvious single characters. For example, one is '1', big_a is 'A', and little_a is 'a'. The list should be used as if these obvious items were included. Note that not every letter of the alphabet is included (B, E, O, U, V, and W are omitted).

absolute_value = 'abs'
and = 'and'
arc_cosine = 'acos'
arc_sine = 'asin'
arc_tangent = 'atan'
block_delete = '/'
cosine = 'cos'
decimal_point = '.'
divided_by = '/'
equal_sign = '='
exclusive_or = 'xor'
e_raised_to = 'exp'
end_of_line = ' ' (non-printable newline character)
fix_down = 'fix'
fix_up = 'fup'
left_bracket = '['
left_parenthesis = '('
minus = '-'
modulo = 'mod'
natural_log_of = 'ln'
non_exclusive_or = 'or'
parameter_sign = '#'
plus = '+'
power = '**'
right_bracket = ']'
right_parenthesis = ')'
round = 'round'
sine = 'sin'
space = ' ' (non-printable space character)
square_root = 'sqrt'
tab = ' ' (non-printable tab character)
tangent = 'tan'
times = '*'

Appendix G Setup File Format

The format of a setup file is shown in Table 10. A setup file is used by giving its name when prompted to do so when the interpreter starts up, as described in Section 4.1.

The file consists of any number of header lines, followed by one blank line, followed by any number of lines of data. The interpreter just skips over the header lines. It is important that there be exactly one blank line (with no spaces or tabs, even) before the data. The header line shown in Table 10 describes the data columns, so it is suggested (but not required) that that line always be included in the header.

The interpreter reads only the first two columns of the table. The third column, “Other Possible Values,” is included here for information.

Each line of the file contains the name of an attribute in the first column and the value to which that attribute should be set in the second column. Attribute names must be spelled exactly as shown in Table 10 in lower case letters. Where the value is shown in upper case letters in the table, upper case letters must be used, and the alternative values are also required to be in upper case letters. The same attribute name should not be used twice, but the interpreter does not check that. If any attribute name not given in the table is used, an error will result.

The lines do not have to be in any particular order. Switching the order of lines has no effect on the interpreter or on how any NC program will be executed (unless the same attribute is used on two or more lines, which should not normally be done, in which case the data for only the last such line will persist).

Attribute	Value	Other Possible Values
axis_offset_a	0.0	any real number
axis_offset_c	0.0	any real number
axis_offset_x	0.0	any real number
axis_offset_y	0.0	any real number
axis_offset_z	0.0	any real number
block_delete	ON	OFF
coordinate_system	1	2,3,4,5,6,7,8,9
current_a	0.0	any real number
current_c	0.0	any real number
current_x	0.0	any real number
current_y	0.0	any real number
current_z	0.0	any real number
cutter_radius_comp	OFF	LEFT, RIGHT
cycle_i	0.1	any positive real number
cycle_j	0.1	any positive real number
cycle_k	0.1	any positive real number
cycle_l	3	any unsigned integer
cycle_p	0.1	any positive real number
cycle_q	0.1	any positive real number
cycle_r	0.0	any real number
cycle_z	0.0	any real number not less than cycle_r
distance_mode	ABSOLUTE	INCREMENTAL
feed_mode	PER_MINUTE	INVERSE_TIME
feed_rate	5.0	any positive real number
flood	OFF	ON
length_units	MILLIMETERS	INCHES
mist	OFF	ON
motion_mode	80	0,1,2,3,81,82,83,84,85,86,97,88,89
plane	XY	YZ, ZX
origin_offset_a	0.0	any real number
origin_offset_c	0.0	any real number
origin_offset_x	0.0	any real number
origin_offset_y	0.0	any real number
origin_offset_z	0.0	any real number
slot_for_length_offset	1	any unsigned integer less than 69
slot_for_radius_comp	1	any unsigned integer less than 69
slot_in_use	1	any unsigned integer less than 69
slot_selected	1	any unsigned integer less than 69
speed_feed_mode	INDEPENDENT	SYNCHED
spindle_speed	1000.0	any non-negative real number
spindle_turning	STOPPED	CLOCKWISE, COUNTERCLOCKWISE
tool_length_offset	0.0	any non-negative real number
traverse_rate	199.0	any positive real number

Table 10. Sample Setup File

Appendix H Tool File Format

The format of a tool file is shown in Table 11. A tool file is used by giving its name when prompted to do so when the interpreter starts up, as described in Section 4.1.

The file consists of any number of header lines, followed by one blank line, followed by any number of lines of data. The interpreter just skips over the header lines. It is important that there be exactly one blank line (with no spaces or tabs, even) before the data. The header line shown in Table 11 describes the data columns, so it is suggested (but not required) that that line always be included in the header.

Each data line of the file contains the data for one tool. Each line has six entries, the first four of which are required, and the last two of which are optional. It makes reading easier if the entries are arranged in columns, as shown in the table, but the only format requirement is that there be at least one space or tab after each of the first four entries on a line. The meanings of the columns and the type of data to be put in each are as follows.

The “POCKET” column contains an unsigned integer which represents the pocket number (slot number) of the tool changer pocket in which the tool is placed. The entries in this column must all be different.

The “FMS” column contains an unsigned integer which represents a code number for the tool. The user may use any code for any tool, as long as the codes are unsigned integers.

The “TLO” column contains a real number which represents the tool length offset. This number will be used if tool length offsets are being used and this pocket is selected. This is normally a positive real number, but it may be zero or any other number if it is never to be used.

The “DIAM” column contains a real number which represents the diameter of the tool. This number is used only if tool diameter compensation is turned on using this pocket. This is normally a positive real number, but it may be zero or any other number if it is never to be used.

The “HOLDER” column may optionally be used to describe the tool holder. Any type of description is OK. This column is for the benefit of human readers only.

The “TOOL DESCRIPTION” column may optionally be used to describe the tool. Any type of description is OK. This column is for the benefit of human readers only.

The interpreter only reads data from the first four columns of each line. The rest of the line is read but ignored.

The units used for the length and diameter of the tool may be in either millimeters or inches, but if the data is used by an NC program, the program must call out the correct G code (G20 or G21) for those units before the data is used. The table shows a mixture of types of units.

The lines do not have to be in any particular order. Switching the order of lines has no effect on the interpreter or on how any NC program will be executed (unless the same slot number is used on two or more lines, which should not normally be done, in which case the data for only the last such line will persist).

POCKET	FMS	TLO	DIAM	HOLDER	TOOL DESCRIPTION
1	1	1.0	0.25	14141	0.25 inch end mill
20	1419	4.299	1.0	0	1 inch carbide end mill
21	1025	8.34	0.5	drill chuck	1/2 inch spot drill short
32	1764	296.515	8.5	0	8.5 mm drill
41	1237	228.360	10.0	0	10 mm x 1.25 tap
60	71117	0	0	0	large chuck

Table 11. Sample Tool File