Flexible automatic converting of nc programs

Thomas Schroeder, Michael Hoffmann

Flexible automatic converting of nc programs

Journal:	*International Journal of Production Research*
Manuscript ID:	TPRS-2005-IJPR-0003.R1
Manuscript Type:	Original Manuscript
Date Submitted by the Author:	24-Aug-2005
Complete List of Authors:	Schroeder, Thomas; IWU Chemnitz Hoffmann, Michael; IWU Chemnitz
Keywords:	CNC, CONTROL, MACHINE TOOLS
Keywords (user):	RS-274, Convert

International Journal of Production Research
Vol. 00, No. 00, 15 July 2004, 1–10

International Journal of Production Research: Flexible automatic converting of nc programs

A cross-compiler for structured text

T. Schroeder[1,2], M. Hoffmann[3]

(00. Month 2005)

The syntax and semantics of programs for CNC machines is defined in standards of the International Organization for Standardization (ISO) and in local counterparts. Because of the very tight limits of these standards, most manufacturers of numeric controls extend the NC language by including specific structures. These could be simple commands, management of variables, control structures etc.

By using the above mentioned manufacturer-specific commands, a NC program can not be used on a numeric control of another manufacturer. Sometimes, even the manufacturer-specific commands differ from newer software versions on the same control.

The present algorithms can be used to support the user by converting NC programs from one type to another. The main focus was set up on on versatility and simple usage. A software tool was programmed to verify the benefits of this approach.

1 Introduction

In general, the programing of CNC machines is done by NC programs written in the RS-274 machine tool programming language. The syntax as well as the semantics are defined in ISO (1982). These standards are not sufficient to implement complex operations. Therefore, most manufacturers of numeric controls extend the NC language by including specific structures. The range of these specific structures reach from simple commands to complex control structures similar to standard languages for computer programing like C or PASCAL.

Therefore, existing parsers like the NIST RS274NGC Interpreter (Frederick et al. (2000)) can only decode NC programs written in pure RS-274 code. Programs which use all features of a given control can be parsed only partly.

[1]Fraunhofer Institute for Machine Tools and Forming Technology IWU, Reichenhainer Str. 88, 09126 Chemnitz, Germany; tel: +49-371-5397-405; fax: +49-371-5397-6405.
[2]To whom correspondence should be addressed. e-mail: thomas.schroeder.tu-chemnitz@iwu.fhg.de
[3]Fraunhofer Institute for Machine Tools and Forming Technology IWU, Reichenhainer Str. 88, 09126 Chemnitz, Germany; tel: +49-371-5397-108; fax: +49-371-5397-6108.

NC programs which include such specific commands can not be used on controls of different manufacturers. Consequently, producers and users of CNC machines have to carry on using controls of one manufacturer even if better (or cheaper) controls are available.

Converting NC programs by human take very long because the differences between the NC dialects are significant. Furthermore, it is difficult to keep concentrated for 50'000 lines of NC code and more. Therefore manufacturers of machine tools often use only one single type of controls. If, for any reason, the controls have to change, all existing NC programs have to be converted manually or completly written new.

Such tasks can be fulfilled by CAD/CAM-systems as far as possible automatically. However, such systems have often a very limited and not expandable language scope. This leads to the fact that some numerical control programs are not produced by CAD/CAM-systems, but written or modified manually to benefit most of the advantages of the installed control system. In addition CAD/CAM-systems are very expensive and therefore usually not affordable for many smaller companies. In other words, there is a multiplicity at numerical control programs which are created manually and/or generated by own software systems.

In this article an different approach is presented. Because of the defined structure of NC programs it is possible to do a lot of conversion work automatically. By using state-of-the-art techniques of computer science, a tool was developed to support the user by converting NC programs from one language to another. The priority was set up on versatility for working on different controls. The tool should be simple enough to be enhanced and modified by the user. It was planned to start with a simple knowledge base and expand it step by step.

2 Concept

NC programs are text files in which the biggest semantic structure is a line. These lines are called "NC blocks". Although there are semantic structure in the manufacturer-specific commands which can be longer than one line, the converting tool should work on the NC program sequentially for every line. In most cases the long commands can be subdivided into parts which can be converted separately.

By converting each line sequentially, the problem of conditional branches is eluded. If any algorithm should proceed through the program in the exact same manner as a real control does, it have to evaluate the condition of all conditional branches. But these conditions are often related to the process. The problem gets even worse when programms for monitoring are designed as

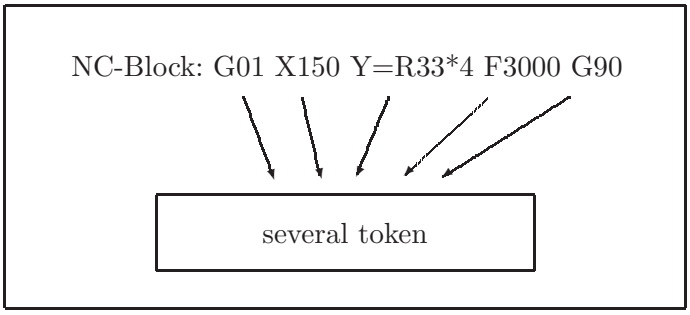

Figure 1. NC block and token

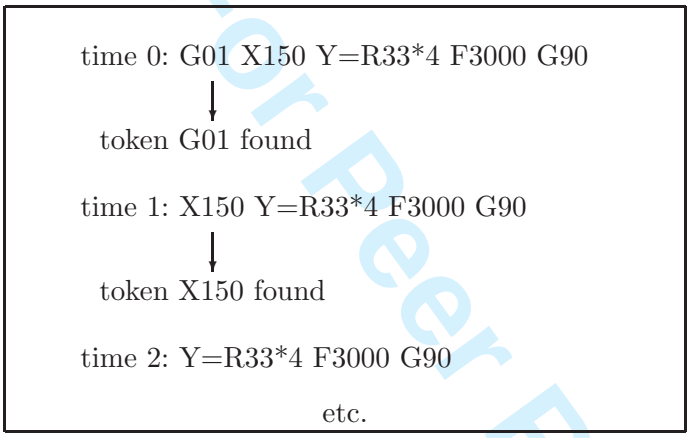

Figure 2. Searching for token at begin of line and deleting them

infinite loops. Because of that, the presented algorithm does not follow any branches, but converts each line sequentially.

Each NC block consists of some token (Fig. 1). If a token consists only of a letter and a number like `"X100"` or `"X=100"` this token is called "NC word". In this work, a token could also something completely different. The `"IF"` of an `"IF-THEN-ELSE"` construct, for example, is also a semantic token.

In a NC block could be any number of token so that the blocks have to be divided into semantic parts. Regular expressions are used to identify these parts. To simplify the expressions, only the beginning of the actual line is checked. If a token is identified its part of the line is deleted (Fig. 2) and the token itself is kept in memory. So the user has only to think of the border that is behind the token. The fore border is always the begin of the line.

By deleting each identified part of the NC block, it is ensured the no character string of this block is found more than once. This reduces the chance of wrong identifications.

A problem occurs when none regular expression of any token matches. In

most cases, the current token has not been defined. Sometimes however, a token was identified wrongly because of a bad regular expression. In both cases, the software tool shows a message and tries to convert the line as far as possible. So the user is warned and could check the specified line. By using this approach, the chance of identifying and converting a token wrongly is minimized. When the line is empty, all token have been correctly identified.

After dividing the line, there are some semantic token in memory. So, after parsing the line, the converting process can be started. The work is done by pre-defined converting operations. The user has to specify a number of operations for each token to convert the semantic of this token. These informations are saved together with the regular expressions in a extern XML file.

3 Regular Expressions

The mathematician S.C. Kleene developed a formal syntax called "regular sets" to describe state machines in computer science (Kleene (1956)). Later, this syntax became a standard called "regular expressions" and was widely used on unix for describing search patterns.

The regular expressions are complex enough to describe the grammar of any regular language. Therefore, this description standard is a type 3 grammar in the Chomsky hierarchy (Chomsky (1956,/)). So the morphology of all natural languages as well as all computer languages can be described by regular expressions. It is complex enough to identify any construct of the RS-274 machine tool programming language.

There are parser generators like YACC (*IEEE POSIX P1003.2* (1992)) for generating parser code of a given grammar written in BNF notation. The main drawback of this approach is the prerequisite to define the complete language. To enhance or change the abilities of the parser, new code have to be generated and compiled. Therefore a change of the convert program binary will be necessary.

In the represented software tool, regular expressions are used to identify semantic token. For example, the expression to identify a G0 command could look like `"G0*0(\s|[[:alpha:]]|$|;)"`. Due to this fact, the user is able to define regular expressions for searched tokens on runtime. The convert program binary can be left unchanged, all modifications are done in a external rules file.

The user may start with a small rules file for the main syntax of the source and target control. Later on, when using the tool to convert real programs, the user can enhance and adapt the rules by simply change the XML rules file. There is no need to alter the algorithm or the binary. Therefore, the tool can also be used for each combination of source and target controls.

Additionally to the regular expression, a special context is required for some token. For example, if `"X="` is written in NC code, a context for the following string is set. On order to search for token, context conditions can be set within the XML rules file.

4 XML Rules File

Extensible Markup Language (XML) (Yergeau et al. (2004)) is a simple, flexible text format that derives from Standard Generalized Markup Language (SGML) (WWW Consortium (1986)). There are many applications for working with XML. Most of them are used in connection with Internet related processes like CSS and HTML.

In this project, XML is used to structure the extern conversion rule file. The user can edit the rules file with any XML editor or just change the plain text to define new token or change the convert operations for existing token.

Because XSLT processors (Tidwell (2001)), which also use rules files based on XML (XSLT code), require XML tags in the source dokument, they cannot be used easily for converting NC programs. To process a NC program with a XSLT processor, a XML tag have to be set before and after each token. Therefore, in this work, the parsing is implemented by regular expressions and the conversion is realized by special rules.

The rules file is a tree of defined token in XML. Because of the fact that each knot can expect conditions (for variables and context), only these routes have to be traced which are relevant to the current convert situation. For example a NC block starts with the function call `"MSG("`. The following character string may only consist of elements which are allowed as parameters of the message function. Therefore the identification of `"MSG("` as the message function triggers a change of the current context.

When working on mathematical expressions, opening parentheses change the context to a new value. Closing parentheses, however, change the context back. Therefore the information about the current context have to be implemented as a stack memory. There are more examples in the manufacturer-specific commands where this is useful.

Each single token has to be defined in the XML file. Every one of them fills a given XML structure or parts of it. These are the leafs of the XML tree. What a token is in fact, can be decided by the user when defining them. It should be the smallest semantic structure of the source control for a successful conversion. Figure 3 shows the structure of a token definition in the XML rules file.

In the structure for one token, its identification is provided by regular expressions and its conversion is defined by convert operations. A change of the

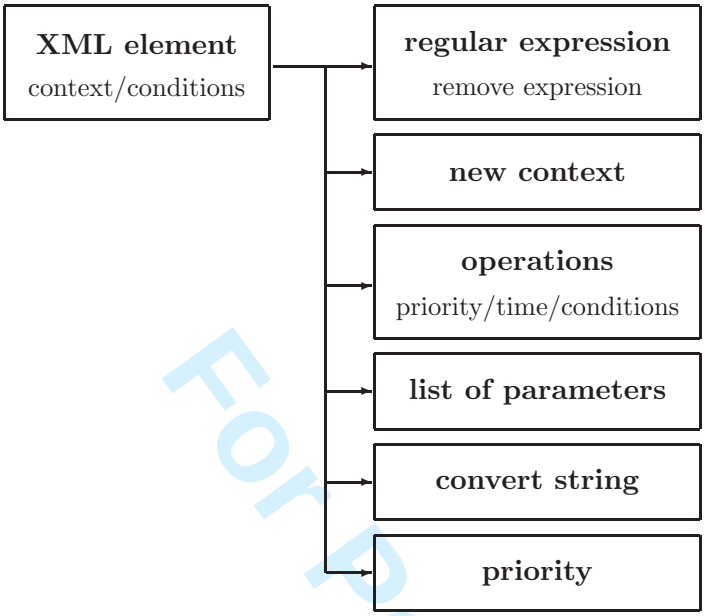

Figure 3. Structure of a token definition

current context can be set as well as a simple conversion string and a list of appendant parameters which are assigned to the token in the conversion process. Finally, a priority can be set. All operations are executed with decreasing priority. Consequently, this also influences the position of the token in the converted NC block.

5 Variables and Operations

The convert operations work on three different types of variables.

- There are local variables which scope is only the token they belong to.
- The scope of global variables is a NC block (one line).
- Eventually there are resistant variables with a scope of the whole NC program.

The user can even call a convert operation which sets a complete token resistant. This is very useful when working on resistant command words.

Conversion variables can be used as simple strings, one dimensional arrays and subsets of parameter token.

The operations mainly change strings. There are also convert operations which work with regular expressions, do NC variables management, specialized operations for modal commands and even operations for writing to disk.

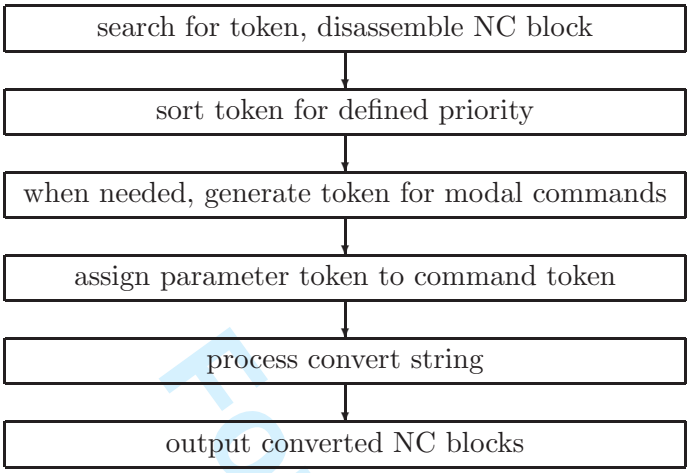

Figure 4. Structure of the conversion process

It could be shown that the program is able to convert each NC block which semantic can be done by n NC blocks ($n \in N$) of the target system. Of course, a comprehensive rules file has to be provided. Even with small rule sets, however, the automatic conversion rate is about 85 percent. If the set of rules is ad

The conversion process has a defined structure (Fig. 4). Between each step, convert operations could be done. The user can specify the time of execution of all operations in the XML rules file.

The conversion program is working on the given plan and executing all operations which have been defined for the found tokens.

One special operation is "ShowWarning". It generates a message for the user. All these messages are written to the output file. The line number of the NC block which caused the message is added to help the user in checking them. The level of a message can be set to `ERROR` or `WARNING` to indicate its importance.

6 Technical Features

Sometimes, the new control works on different axis names. For this reason, a text file can be specified to define old and new axis names. If any of the old names is found, it will be replaced by the assigned new one.

In most controls there are machine specific switches called "help functions". They could be used for switching the cooling or control the tool pallet. They are constant for one machine and they do not have any parameters. So, in this tool the user may define such help functions as simple strings. The search

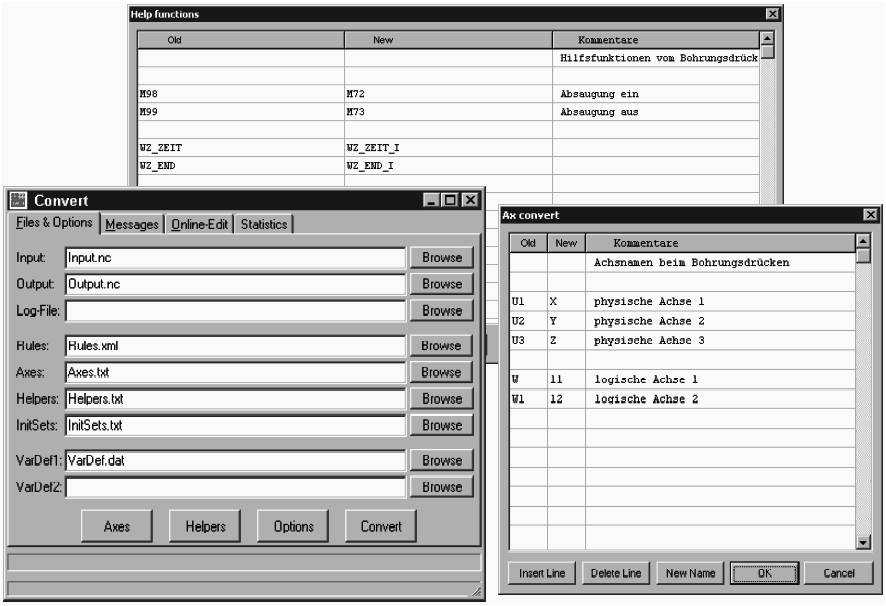

Figure 5. Graphical user interface

for help functions has a higher priority than the search for a matching with regular expressions. If any of these possibilities are found, a token is generated which only consists of the defined target help function. Of course any character string can be defined as help function.

On each control, some modal commands are active on startup. Some NC programs refer to these initialization values. Therefore, the user can provide a text file to setup the initialization values. This way also global variables in NC programms as well as extern functions can be defined. The conversion tool indentifies them correctly and converts them as described in the XML rules file.

Finally a intuitive graphical user interface have been programmed (Fig. 5). Most options can be set in this windows application.

To keep close contact to the user while converting, the program stops optionally if an error occurs. In this case, a window (Fig. 6) is opened where the user can change on-line the generated code. It's also possible to set specified messages not to be shown again or not to be written to the output file.

7 Conclusion

There is a need of a software tool that helps the user of CNC machines to convert NC programs from one manufacturer-specific NC dialect to another in

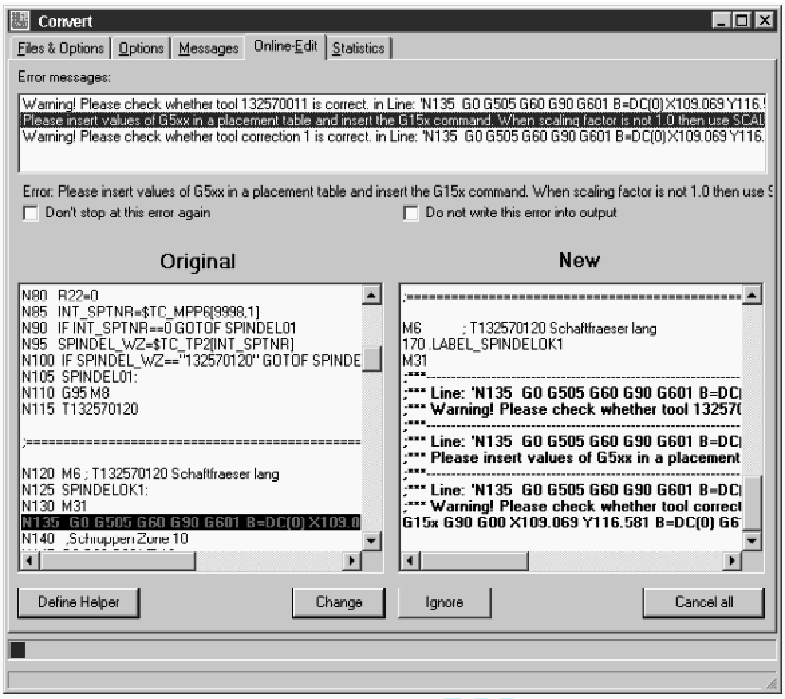

Figure 6. Online edit mode

order to be more flexible concerning the usages of controls. Because the differences between these dialects are significant, the conversion effort is enormous. On the other hand, the NC programs are very structured that most of the work can be done automatically.

The represented software is very flexible by using an extern rules file. The structure of this file is written in XML to have a widely used base. This set of rules can be extended easily by the user. Each defined token enhances the functionality of the converter.

The parsing of the NC code is done by regular expressions so that all computer languages can be parsed. The conversation is executed by provided convert operations that range from simple string editing to NC variable management. In tests this tool translates up to 90 percent of complex NC programs automatically. For the last 10 percent, the user is provided with specific informations about the problem and possible solutions. So the time to get an existing NC program usable on different control is very shortened.

REFERENCES

Chomsky, Noam (1956), 'Three models for the description of language', *IRI Transactions on Information Theory* **2**(3), 113–124.

Chomsky, Noam (1956/1975), *The Logical Structure of Linguistic Theory*, Plenum.

Frederick, Thomas Kramer, Frederick Proctor and Elena Messina (2000), 'The nist rs274ngc interpreter - version 3', National Institute of Standards and Technology. NISTIR 6556.

IEEE POSIX P1003.2 (1992).

ISO (1982), *Numerical control of machines - Program format and definition of adress words*.

Kleene, S. C. (1956), Representation of events in nerve nets and finite automata, *in* 'Automata Studies', Pinceton University Press, pp. 3–42.

Tidwell, Doug (2001), *XSLT*, 1st edn, O'Reilly.

WWW Consortium (1986), *Information processing – Text and office systems – Standard Generalized Markup Language (SGML)*, ISO 8879.

Yergeau, Franois, Eve Maler, Jean Paoli, Tim Bray and C. M. Sperberg-McQueen (2004), *Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C, http://www.w3.org/TR/REC-xml/.