NBSIR 87-3677

ERROR PREVENTION AND DETECTION IN DATA PREPARATION FOR THE VERTICAL WORKSTATION MILLING MACHINE

November 19, 1987

By:
Thomas R. Kramer
Timothy Strayer

CONTENTS

Page

I. INTRODUCTION 1
1. CONTENTS 1
2. AUDIENCE 1
3. BRIEF VWS DESCRIPTION 1
4. DESIGN PROTOCOL 2
5. RELATED READING 3

II. GENERAL APPROACH TO ERROR PREVENTION AND DETECTION 5
1. NEED FOR VERIFICATION AND CURRENT PRACTICE 5
2. VWS2 VERIFICATION SUBSYSTEM 5
2.1. Introduction 5
2.2. Software 7
2.3. Messages from the Verification Subsystem 7
2.4. User Control Over the Verification Subsystem 7
2.5. Simplifying Techniques 7
2.6. Effectiveness of the Verification Subsystem 8

III. DESIGN EDITOR DIALOG 9

IV. DESIGN ENHANCEMENT 11

V. DESIGN VERIFICATION 13
1. INTRODUCTION 13
2. PARAMETER TYPE CHECKING 13
3. FEATURE VERIFICATION 13
4. REFERENCE FEATURE FIT CHECKING 13
4.1. Introduction 13
4.2. Subfeature Considerations 14
4.3. Contour_groove as Reference Feature 16
4.4. Geometry of Reference Feature Checking 18
4.5. Software for Reference Feature Checking 18

VI. PROCESS PLAN VERIFICATION 19
1. INTRODUCTION 19
2. DATA FOR PROCESS PLAN VERIFICATION 19
2.1. Introduction 19
2.2. Milling Area Model 20
2.2.1. General 20
2.2.2. Obstacles 20
2.2.3. Safe_z_plane 20
2.2.4. Automatic Updating of Vise Obstacle Data 22
2.2.5. Machining Parameters 24
2.3. Tool Geometry Model 24

3. CHECKS INCLUDED IN OPERATION VERIFIERS 26
3.1. Introduction 26
3.2. Tool Must Be Suitable 26
3.2.1. Center_drill 26
3.2.2. Counterbore 26
3.2.3. Drill_hole 26
3.2.4. Face_mill 26
3.2.5. Fly_cut 26
3.2.6. Machine_chamfer_in 26
3.2.7. Machine_chamfer_out 27
3.2.8. Machine_countersink 27
3.2.9. Mill_contour_groove 27
3.2.10. Mill_contour_pocket 27
3.2.11. Mill_groove 27
3.2.12. Mill_pocket 27
3.2.13. Mill_side_contour 27
3.2.14. Mill_straight_groove 27
3.2.15. Mill_text 28
3.2.16. Set0_center, Set0_corner, and Set0_z 28
3.2.17. Tap_thread 28
3.3. Tool Holder Must Stay Above Top of Part 28
3.4. Shank Must Clear Reference Features 28
3.5. Do Not Cut Deeper than Flutes 28
3.6. Tool Tip Must Clear Workpiece 29
3.7. Stay Within Part or Machining Box 31
3.7.1. On the Sides 31
3.7.2. On the Bottom 31
3.8. Do Not Hit Obstacles 32
3.8.1. Side Obstacles 32
3.8.2. Bottom Obstacles 32
3.9. Spindle Speed Must Be in Reasonable Range 33
3.10. Feed Rate Must Not Be Too High 33
3.11. Pass Depth Must Be in Reasonable Range 33
3.12. Additional Tests 34
3.12.1. Center_drill 34
3.12.2. Face_mill and Fly_cut 34
3.12.3. Mill_contour_pocket 34
3.12.4. Mill_pocket 34
3.12.5. Mill_side_contour 34
3.12.6. Set0_corner 34
3.12.7. Tap_thread 35
4. SOFTWARE FOR MACHINING OPERATION VERIFICATION 35

VII. WORKPIECE VERIFICATION....37

VIII. PART MODEL CHECKING....39

IX. OTHER AUTOMATIC VERIFICATION....41
1. GENERAL....41
2. INITIATION OF THE DATA EXECUTION MODULE....41
2.1. Introduction....41
2.2. Design Checks....41
2.3. Process Plan Checks....41
2.4. Workpiece Checks....42
2.5. Fixturing Checks and Machining Checks....42

X. DESIGN DRAWING....43

XI. WORKPIECE MODEL DRAWING....45

XII. TOOL PATH DRAWING....47

XIII. AUTOMATIC GENERATION OF FEATURE VERIFIERS....51
1. INTRODUCTION....51
2. HOW THE AUTOMATIC PROGRAMMING SYSTEM WORKS....53
3. FORMAT FOR RULES....53
4. AN EXAMPLE....56
5. AUTOMATIC PROGRAMMING SYSTEM SOFTWARE....57
6. STRENGTHS AND WEAKNESSES....57

XIV. LIMITATIONS....59
1. INTRODUCTION....59
2. PHYSICAL OBJECT DATA NOT CHECKED....59
3. MILLING AREA SIMPLIFICATIONS....59
4. TOOL GEOMETRY SIMPLIFICATIONS....59
5. TWO-AND-A-HALF-DIMENSIONAL MODELLING....59
6. PRE-ENHANCEMENT DESIGN VERIFICATION NOT DONE....59

REFERENCES....60

LIST OF FIGURES

Page

Figure 1. Reference Feature Fit Checking 15
Figure 2. Contour Grooves As Reference Features 17
Figure 3. Milling Area Model 21
Figure 4. Updating Vise Obstacles 23
Figure 5. Tool Geometry Model 25
Figure 6. Tools Hitting Reference Features 30
Figure 7. Tool Path Drawing 49
Figure 8. Feature Verification Function 52

LIST OF TABLES

Page

Table 1. Verification Summary.....6
Table 2. Part Design Editor Dialog.....10
Table 3. BNF Definition of an Acceptable Verification Rule 55

ERROR PREVENTION AND DETECTION IN DATA PREPARATION FOR THE VERTICAL WORKSTATION MILLING MACHINE IN THE AUTOMATED MANUFACTURING RESEARCH FACILITY AT THE NATIONAL BUREAU OF STANDARDS

Dr. Thomas R. Kramer
Guest Worker, National Bureau of Standards, &
Research Associate, Catholic University

Mr. W. Timothy Strayer
Computer Scientist, National Bureau of Standards

November 19, 1987

Funding for the research performed by Dr. Kramer and reported in this paper was provided to Catholic University under Grant No. 60NANB5D0522 and Grant No. 70NANB7H0716 from the National Bureau of Standards.

Certain commercial equipment and software are identified in this paper in order to adequately specify the experimental facility. Such identification does not imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the equipment and software identified are necessarily the best available for the purpose.

ERROR PREVENTION AND DETECTION IN DATA PREPARATION FOR THE VERTICAL WORKSTATION MILLING MACHINE IN THE AUTOMATED MANUFACTURING RESEARCH FACILITY AT THE NATIONAL BUREAU OF STANDARDS

I. INTRODUCTION

1. CONTENTS

This paper discusses error prevention and detection in data preparation for the milling machine in the Vertical Workstation (VWS) of the Automated Manufacturing Research Facility (AMRF) at the National Bureau of Standards. The descriptions pertain to the system in use during the summer of 1987.

Chapter II presents the overall approach used for error prevention and detection, Chapters III through XII discuss ten types of verification included in the system, Chapter XIII presents the rule-based automatic programming subsystem which is used to write the LISP functions which verify features, and Chapter XIV gives a number of limitations of the system in error prevention and detection.

2. AUDIENCE

The paper is intended to be useful to people interested in concepts and technical details of the VWS, particularly AMRF personnel who are running the VWS or maintaining or improving the software for the VWS. The paper is intended to be useful also to other researchers in automated manufacturing. A knowledge of the computer language LISP is useful but not essential to reading this paper. Detailed documentation of the LISP functions that are involved with the systems described here is being prepared separately.

3. BRIEF VWS DESCRIPTION

The VWS is a computer-integrated automated machining workstation. It includes a control system, a computer-aided design system, an automatic process planning system, and an automatic nc-code generator. The principal machinery is a milling center (Monarch VMC-75 with a GE2000 controller) and a robot (Unimate 4070 with a Val II controller) to tend the milling center. There is quite a bit of ancillary hardware. The system is controlled from a microcomputer (Sun 3/160 with 6M memory, BW monitor). Running in stand-alone mode, it is possible to design and machine a simple metal part within an hour. The VWS may also be run as an integrated part of the AMRF. The workstation is described in more detail in [K&J1].

The software for the VWS is written in the computer language LISP. In this paper this software is called the VWS2 system. The error prevention and detection routines included in the VWS2 system are referred to in this paper as the verification subsystem.

Six principal modules comprise the VWS2 system: the Production Management Operating System (the control system), the State Table Editor, the Equipment Program Generator, the Part Design Editor, the Process Planner, and the Data Execution module.

The Part Design Editor, Process Planning and Data Execution modules, as well as other system capabilities, may be accessed by the user through a small user interface called vws_cadm. Vws_cadm asks the user questions about what the user wants to do and then activates the appropriate module or other capability accordingly.

To produce a part from scratch, the user sits at the Sun workstation and creates a design using the Part Design Editor. The Process Planner is then called to write a plan for how to machine a part of that design. Next NC-code is generated automatically from the design and the plan by the Data Execution module. Finally the user tells the control system to make the part. The control system coordinates the activities of the workstation equipment so that the part blank is loaded onto the milling machine, the NC-code is sent to the milling machine and executed (making the part), and the finished part is unloaded.

This paper discusses error prevention and detection in those parts of the system that involve data preparation for the milling machine, namely, the Part Design Editor, Process Planning, and Data Execution modules.

4. DESIGN PROTOCOL

The VWS2 system uses a feature-based design protocol. The design protocol is described in detail in [K&J2]. The design of a part is expressed as a list of features on a piece of stock. The piece of stock is always a rectangular block. The design protocol currently assumes that all features are being made from one side of the block.

Although all the features and subfeatures are purely geometric, they were selected to be included in the system on the basis of being features commonly found on machined parts that could be produced in one, or at most a very few, machining operations. Each feature and subfeature is a removed volume.

The primary features in the system in September, 1987 are: chamfer_out, groove, hole, pocket, straight_groove, text, contour_groove, contour_pocket, and side_contour. There are also subfeatures which may be made on the primary features: chamfer_out, chamfer_in, countersink, and thread. A feature is specified in the system by giving its name and the values of several parameters which specify its location, shape, and size.

The design protocol includes the use of “reference features”. If feature A is to be made at the bottom of feature B, then one of the parameters of feature A is “reference_feature”, and the value of that parameter is the feature number of feature B. Whenever B is the reference feature for A, the outline of feature A must fit within the outline of feature B, and the bottom of feature B must be flat (except in the case of concentric drill holes).

5. RELATED READING

A more succinct discussion of the verification subsystem is given in [KR&S]. A separate verification subsystem for the VWS Equipment Program Generator has been built by Kie Nakpalohpo and is described in [NA&J].

This paper is one of about a dozen papers being prepared as part of the AMRF documentation to describe all aspects of the VWS. The others are [JUN], [KRA1], [KRA3], [KRA4], [KRA5], [K&J2], [K&J3], [KR&W], [LOVE], [RUDD]. Background information about the VWS2 system needed to understand the verification subsystem in detail is not included in this paper. To get enough background concerning the VWS2 system modules, the design protocol, and the process plan protocol to understand the verification subsystem fully, a reading of [K&J1] should be sufficient.

II. GENERAL APPROACH TO ERROR PREVENTION AND DETECTION

1. NEED FOR VERIFICATION AND CURRENT PRACTICE

In using an NC milling machine the consequences of error can be very costly. One incorrect character in an NC program of 10,000 characters (which is a moderate size) can break a tool, ruin a workpiece, cripple the machine, or worse. It is very unusual for a human to prepare an error free file 10,000 characters long in any language. Since NC-code is usually prepared with human involvement (although current systems vary from heavily manual to largely machine-aided), it is common practice in machine shops to check out NC programs before they are used to cut a finished part.

On a milling machine, checking may take the form of cutting air, cutting a soft material such as styrofoam, or cutting at reduced speed, with an operator watching closely all the time and keeping a hand on the stop button. Some computer aided NC-coding systems include a graphical NC-code checking subsystem. Typically, the display of such a subsystem will show the workpiece and its fixturing in a three dimensional picture. The actual NC code will be fed into the system, and a picture of the swept volume generated by the tool as the code is executed will be shown. The user may then judge if the tool is cutting properly and safely. All of these checking methods require the time of a human worker, time on the milling machine, or both. This is very expensive.

The cost of producing the first actual part of a given design is high for several additional reasons (process planning and setup costs, especially), but the cost of checking is usually a significant factor.

2. VWS2 VERIFICATION SUBSYSTEM

2.1. Introduction

In the VWS2 system we have tried to automate whenever feasible, including automatic verification. The kinds of verification provided by the system are summarized in Table 1. We will refer to the types of verification listed in Table 1 as “verification components”. Much of the verification is automatic, but three types are human-interactive. Once the design phase is complete, no human checking is required (although is it possible). A user who knows the system can make checks in addition to those of Table 1 by interrupting the normal flow of data and reading the process plans and NC-code produced by the system.

Table 1. Verification Summary

TYPE OF VERIFICATION	WHERE USED IN THE VWS2 SYSTEM			
	Design Editor Module	**Process Planning Module**	**Data Execution Module**	**Directly from vws_cadm User Interface**
AUTOMATIC				
Design Editor Dialog	**X**			
Design Enhancement	**X**	**X**	**X**	**X**
Design Verification	**X**		**X**	**X**
Process Plan Verification			**X**	
Workpiece Verification			**X**	**X**
Part Model Checking			**X**	
Other	**X**	**X**	**X**	**X**
USER INTERACTIVE Design Drawing	**X**			**X**
Workpiece Model Drawing			**X**	
Tool Path Drawing				**X**

2.2. Software

The verification subsystem is the largest VWS2 subsystem. Roughly a quarter of the 700 or so pages of code written by the authors for the VWS2 system is used only for verification, and another quarter is used for multiple purposes, one of which is verification.

The "verify2" subdirectory of the "vws2" directory contains 71 functions which are devoted exclusively to verification. It includes 18 functions for reference feature fit checking, 13 for feature verification, and 40 for machining operation checking. The "vergen2" subdirectory contains 12 functions which generate feature verification functions, as well as 9 files of rules written in constrained English. These four sets of LISP functions are described individually in the remainder of this paper.

All 65 functions of the "geom2" subdirectory [K&J2] are also required by the verification subsystem. Dozens of functions which serve other primary purposes make a check or two.

2.3. Messages from the Verification Subsystem

Our approach to dealing with an error that is detected is to notify the user as specifically as possible what the error is. The system includes roughly 400 distinct error messages, not counting dynamically generated variations, such as inserting a feature or step number. As a rule, error messages are complete, grammatically correct English sentences, usually one or two lines long. The majority of the messages are contained in the design and process plan verification components.

2.4. User Control Over the Verification Subsystem

Some of the verification subsystem is at work whenever the system is being used, but the design and process plan verification components are under user control. In the Part Design Editor module, the Data Execution module, and some facilities available directly from the user interface, the user may set the verification mode to "off", "on soft", or "on hard". In the Design Editor, if verification is on hard and an error is encountered, the system will refuse to have anything more to do with the feature that caused the error. In the Data Execution module, the module quits completely if an error is encountered while verification is on hard. If verification is on soft and an error is encountered, the system will ask the user if it should attempt to continue at risk of further error. If verification is off, the design and process plan verification components will not be used.

2.5. Simplifying Techniques

The verification subsystem includes many simplifying techniques. For example, a piece of fixturing is assumed to have a rectangular or circular profile. This means that some of the verification is approximate. In all cases the simplifications are selected so that some features or operations which are OK may be reported as errors, but no feature or operation which is not OK will fail to be reported in error.

2.6. Effectiveness of the Verification Subsystem

The verification subsystem has proved to be very effective. For one-sided parts, it is a routine matter to have the VWS2 system automatically generate a process plan and an NC program 10,000 characters long to produce a newly designed part, with no errors. The part that is produced is exactly as designed, within the tolerances provided by the operation of the milling machine.

III. **DESIGN EDITOR DIALOG**

A part design could, in principle, be created by using a text editor. A design made this way would undoubtedly be full of errors. There would be some typos in the spelling of attributes, some attributes would be forgotten, some values would be the wrong type of data, and so forth. The use of the Design Editor prevents many errors of this sort in three ways.

First, errors in attributes never arise because the names of all attributes for each feature are in the LISP code of the Editor, and the Editor always insists that the user provide values for all that are required, and always spells the names correctly.

Second, many errors in values are prevented in one of two ways: either the Editor requires the user to choose from a list of acceptable values, or the Editor does some checking after the user has entered a value through the keyboard. For example, if the user enters anything except an integer that is already a feature number for the value of a reference feature, the Editor will not accept it. The type of checking that is done is tailored to the particular situation.

Third, the structure of a dialog may be varied according to the user's input. For example, if the user says the depth of a hole should be "thru", the Editor will not ask for a bottom_type.

A sample dialog follows in Table 2. The user has made several errors which the system fends off. User input is shown in boldface, with errors italicized. Everything else is from the Design Editor. The dialog occurs in one window of the computer terminal, while long lists (shown in boxed text in Table 2) are printed by the Editor to a second window.

Table 2. Part Design Editor Dialog

User input is shown in boldface. User errors are in bolface italics. Other text is from the system.

```
pde > i
Enter number of existing feature before which to insert new feature,
or 999 to insert at end. ? ABC
Enter number of existing feature before which to insert new feature,
or 999 to insert at end. ? 999
        1.-  groove
        2.-  straight_groove
        3.-  contour_groove
        4.-  pocket_corners
        5.-  pocket_center
        6.-  contour_pocket
        7.-  side_contour
        8.-  text
        9.-  hole
        10.-  chamfer_out
 choose a number, 0 to ignore ? 12
 illegal selection - try again? 9
 Enter  "center_x"  (numeral) ? 3.a
 Enter  "center_x"  (numeral) ? 3.2
 Enter  "center_y"  (numeral) ? "hi"
 Enter  "center_y"  (numeral) ? 2
 Enter  "diameter"  (numeral) ? .5
 Enter  "depth"  (numeral/thru) ? thro
 Enter  "depth"  (numeral/thru) ? thru
 Adding threads to this hole (y/n) ? n
 Enter "countersink_diameter" (numeral/n) ? .6
 Enter "chamfer_in_depth" (numeral/d(efault)=0.046875/n) ? n
 Enter "reference_feature" (numeral/n) ? 3
 That reference_feature does not exist.
 Enter "reference_feature" (numeral/n) ? 1
 Feature i999 hole is OK.
        feature  2 - hole
              center_x = 3.2
              center_y = 2
              diameter = 0.5
              depth = thru
              bottom_type = nil
              countersink_diameter = 0.6
              reference_feature = 1
pde >
```

IV. **DESIGN ENHANCEMENT**

The Design Enhancement subsystem adds information to the description of each feature in a design. If the feature is a straight_groove which has a chamfer_in_depth, for example, the coordinates of the upper left and lower right corners of an imaginary box around the straight_groove are calculated and added to the description of the groove in the enhanced design. In many cases, tests are made on feature parameters in conjunction with generating this additional information. For a straight_groove, a check is made of whether the x1 and x2 values are equal or the y1 and y2 values are equal (i.e. whether the feature is horizontal or vertical). If not, the feature cannot be chamfered, so the system will not try to generate the corner coordinates. Instead, the system sends a message to the user at the workstation about the problem.

For some features (all three contour features, for example), the amount of information generated by the enhancement subsystem is much larger than the amount provided by the user in defining the feature, and the amount of checking done may be quite extensive.

As a rule, checks are made during enhancement if (1) not checking may produce a dangerous situation (as in the case of the chamfer of a straight_groove where the system might attempt to chamfer in the wrong place) or (2) bad data makes it impossible to perform a calculation.

V. **DESIGN VERIFICATION**

1. INTRODUCTION

Design verification includes three distinct subcomponents: parameter type checking, feature verification, and reference feature fit checking, all three of which are put to work by applying the function “verify_feature” to all the features that need to be verified. If a feature which is given to “verify_feature” has not been enhanced yet, “verify_feature” will use the enhancement subsystem to enhance the feature before doing the other checks.

2. PARAMETER TYPE CHECKING

The “features” data base contains a list of all the parameters which may be used to describe each feature type. Along with each parameter is a list of checks to apply to the parameter. For example, the depth parameter must always be a number (after enhancement), and it must be positive. When parameter type checking is performed on a feature, the system applies all the listed checks to each parameter of the feature. The functions which perform the parameter type checks are standard Franz LISP functions.

3. FEATURE VERIFICATION

Each feature type has a verification function that tests whether a feature of that type obeys a set of rules. The number of rules in one of these functions averages about a dozen, varying from one to twenty. When feature verification is performed, all of the rules for the particular feature type are tested. The rules are given in [K&J2].

The feature verifiers were written by a small automatic programming system designed to take a list of rules, in a somewhat restricted specification language, and generate LISP code to check that each rule is observed. Feature verification is the only place in the verification subsystem that functions were automatically generated. A description of the automatic generation of feature verifiers is included in Chapter XIII of this paper.

Verification functions written by the automatic programming system for the nine feature types are in the “verify2” subdirectory of the “vws2” directory. The name of each begins with the prefix “verify_” and ends with the feature name -- “verify_hole”, for example. Four other functions required for feature verification are in the same subdirectory.

4. REFERENCE FEATURE FIT CHECKING

4.1. Introduction

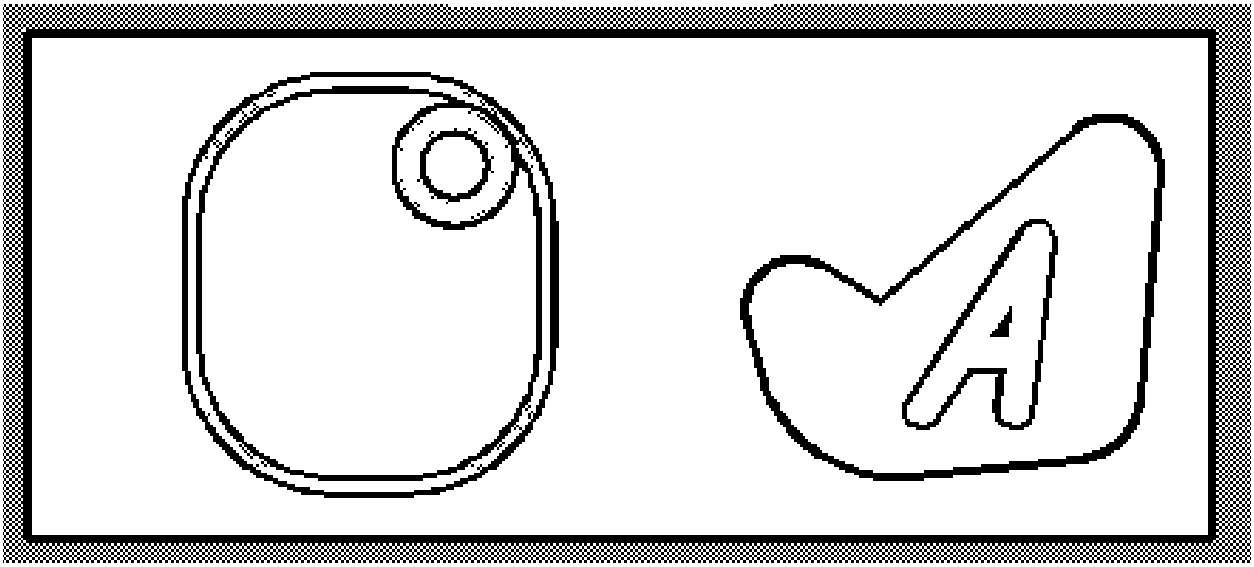
This check comes into play only if a feature (call it Feature A) has a reference feature (call it Feature B). In this case the "verify_feature" function finds the name of a fit checking function in the "features" data base that is specific to the feature types of A and B. The checking function determines whether the outline of feature A will fit inside the outline of feature B. This fit is a requirement of reference features, as described earlier. If the fit requirement is not met, the milling machine might try to move rapidly through metal where it expects to find air. Thus this check is very important in avoiding machining disasters.

In general, the exact outlines of features are used in checking reference feature fit. There are two exceptions to this: (1) if a text feature is to fit inside another feature, and (2) if a contour groove is to be used as a reference feature. For checking text, a parallelogram with rounded corners that just fits around the text, whose top and bottom are parallel to the x-axis, and whose sides are tilted by the tilt factor used for the font is checked. The situation for using a contour_groove as a reference feature is described below. In both cases, some false error messages may be generated, but no unsafe conditions are allowed to slip by.

4.2. Subfeature Considerations

Each fit checking function requires that the particular geometry of both feature types be taken into account, including the full possible range of parameters and subfeatures. For example, if a feature has been chamfered, it will extend beyond its unchamfered outline. When such a feature is checked for fitting within a reference feature, the extended outline must be used for checking. When such feature is checked for whether another feature fits inside it, the unchamfered outline must be used. Examples of a reference feature fitting and not fitting are shown in Figure 1. All three subfeatures (chamfer, countersink, and thread) must be considered.

Figure 1. Reference Feature Fit Checking



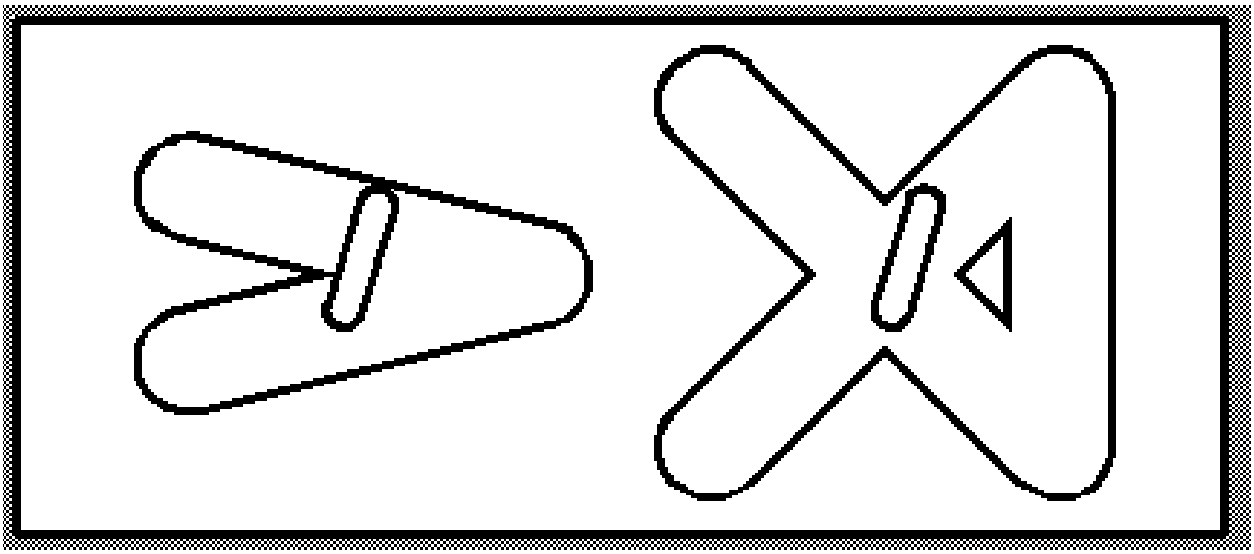
On the left a chamfered pocket is used as the reference feature for a countersunk hole. The hole does not fit in the pocket.

On the right a contour_pocket is used as the reference feature for a text feature. The text fits into the contour_pocket

4.3. Contour groove as Reference Feature

In the case of using a contour groove as a reference feature (i.e. checking whether another feature will fit inside it), a special difficulty arises: the actual outline of the feature is difficult to determine. This is because a contour groove may double back on itself or cross over itself. In the current implementation, the system deals with doubling back in a limited way but does not deal with crossovers. Examples of contour grooves used as reference features are shown in Figure 2.

Figure 2. Contour Grooves as Reference Features



Two contour grooves are shown, each with a straight_groove in it that fits.

On the left, the reference feature fit checking system reports correctly that the straight_groove does fit in the contour_groove.

On the right, the checking system reports incorrectly that the straight_groove does not fit in the contour_groove. The error message results from placing the straight groove at a location where the contour_groove crosses over itself. The system does not realize that a portion of the boundary of the contour_groove has been eliminated.

4.4. Geometry of Reference Feature Checking

The problem to be solved in reference feature checking is whether the outline of one feature fits inside the outline of another, or, sometimes, whether two outlines are disjoint. In some cases there is a simple algorithm to find the answer. For example, circle A fits inside circle B if the distance between the center of A and the center of B, plus the radius of A is less than the radius of B. As another example, a parallelogram with rounded corners fits in a pocket if the circles at the four corners of the parallelogram all fit in the pocket.

The method used for most features, however, is to find the collections of arcs and straight line segments that comprise the two outlines, and to make sure (1) that no line or arc in the first collection intersects any line or arc in the second collection, and (2) that at least one point of the outline which is supposed to be inside is inside. For most features, the collection of arcs and straight line segments which form the outline of the feature is created by constructing a virtual contour outline. Then these virtual contour outlines for the two features can be checked. To check if point A is inside an outline, point B, known to be outside the outline, is selected, and the number of intersections of the line between points A and B with the outline is found. It this number is odd, point A is inside; otherwise point A is outside.

In the case of a contour groove used as a reference feature, since the actual outline of the groove is not sure to be known, a different insideness test is used. A point is inside the outline of a contour groove if it is within half the groove width of one of the arcs and straight line segments that form the center line of the groove.

In order to avoid checking for equality, which is usually not realistic to do with floating point numbers, either the outline of the inner feature is shrunk or the outline of the outer feature is expanded by a number (normally 0.00001 inch) which is small enough to be inconsequential in the real world, but large enough to avoid problems with round-off errors (which occur around the sixteenth significant figure in the decimal representation of a number in LISP).

4.5. Software for Reference Feature Checking

Reference feature checking software is located in the “geom2” and “verify2” subdirectories of the vws2 directory. Almost every function from the geom2 directory (there are 65) is used. Eighteen functions from the verify2 directory are required, the names of which all begin with “ref_”, except for “make_ref_desc”. The top level function is “ref_test”, which takes the descriptions of two features and refers to the “features” data base to find if the second can be used as a reference feature for the first. If so, the correct checking function is selected from the “features” data base and applied to the two features.

VI. **PROCESS PLAN VERIFICATION**

1. INTRODUCTION

The Data Execution module acts by executing a process plan, which is a plan for how to machine a part of a given design. The components of a process plan are the operations needed to produce each feature in the design. The purpose of the operation verifiers is to ensure that each machining operation in the process plan can be executed by the milling machine without making the feature wrong or causing damage to the machine, the fixturing device, the tool, or the part itself. Each operation has its own operation verifier.

The operation verifiers assume that the feature on which an operation is being performed will pass design verification. Items such as reference feature fit are not rechecked by the operation verifiers. This is why the Data Execution module checks the design before checking the process plan. A portion of the data used by the verifiers is extracted from the design. Any aspect of the data taken from the design which has been checked by the design verifiers is not rechecked by the operation verifiers.

Process plan verification is carried out during data execution rather than during process planning for several reasons.

First, a given process plan may be usable on workpieces with different initial geometries. A step of the plan necessary to make a feature already present on the workpiece will be removed from the plan during initialization of the Data Execution module and will never be verified. Thus, what gets verified varies according to the workpiece used.

Second, verification is performed with respect to specific tools in the milling machine, not with respect to a tool_type identifier, and this cannot be done until specific tools are selected by the Data Execution module. One piece of information, the exposed length of a tool, does not exist until the tool is put into a holder. This information cannot be in the tool catalog used for process planning, but is part of the information about the tooling on the milling machine.

Third, fixturing information is not included in the process plan, so testing collisions with fixturing cannot be done until the fixturing information is available.

Fourth and finally, machining parameters are not available during process planning.

2. DATA FOR PROCESS PLAN VERIFICATION

2.1. Introduction

Process plan verification uses knowledge of the milling machine, the tools in the machine, and the fixturing devices on the machine to validate the plan. This knowledge is kept in a data structure called the “world model”. Several machining parameters, such as the minimum thickness to be allowed between a feature and the edge of the original block, are also kept in the world model. There are currently two fixturing devices on the machine: hydraulic pallet clamps and a vise.

When the user selects a fixturing device (one step in starting the Data Execution module) the appropriate fixturing model is activated.

A second critical data source is a model of the geometry of the workpiece at the time when the operation is to be carried out, which takes into account the preceding operations in the process plan. This model is maintained by the Data Execution module.

Additional knowledge, such as the tool path envelope of each cutting algorithm is implicit knowledge assumed by the operation verifiers. The tool path envelope is the surface of the swept volume made by a cutting tool while performing a milling operation, outside of which the tool will never travel. This includes accounting for the cutting radius and shank radius of the tool. The operation verifiers do not attempt to verify NC code; neither do they attempt to verify the tool path that will be used to cut the part. They do verify the tool path envelope. Using tool path envelopes, rather than generating many smaller swept volumes by reading NC code for the tool movements inside the envelope, speeds up calculations enormously without any loss of generality. It is necessary, of course, that the envelope be known correctly.

2.2. Milling Area Model

2.2.1. General

The geometric model of the milling area implicit in the data base and verification subsystem is shown in Figure 3. The milling machine cuts from above the part, which is held fast in a fixturing device. It is assumed that no object in the milling area is above any part of the top surface of the original block. The part is imagined sitting in a topless box whose sides are parallel to the sides of the block. The part is imagined to be floating in the air a distance above the bottom of the box. This distance is called “bottom_clearance”. The bottom and the four vertical sides of the box are maximum run off planes and represent the farthest in the five directions that the tool envelope may be allowed to go.

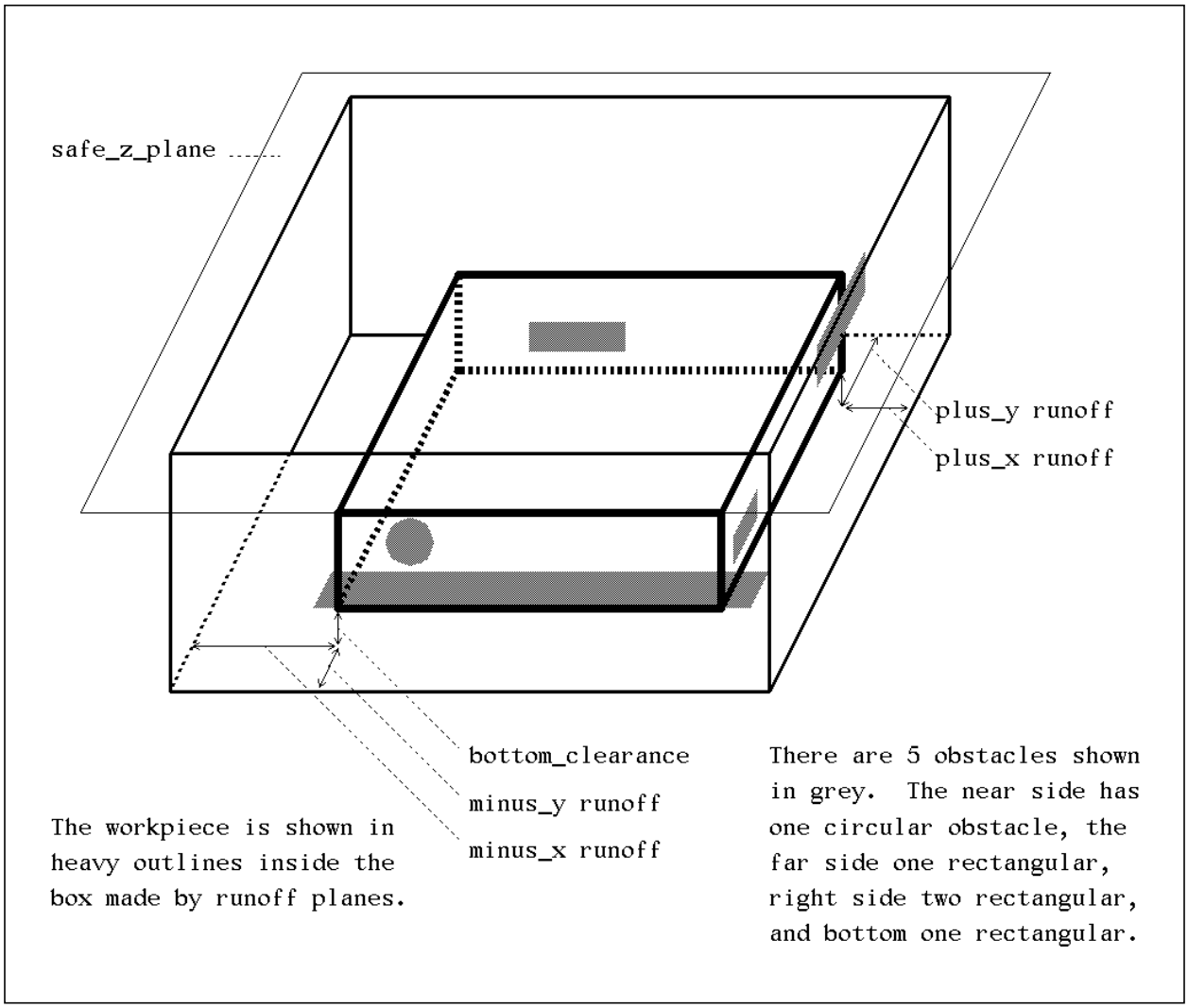
2.2.2. Obstacles

Portions of the fixturing device (or other objects) that are within the box and are not to be milled into are represented as side and bottom obstacles. It is assumed that each side obstacle is in contact with the part (even if not so). Each obstacle is assumed to have a rectangular or circular profile when viewed from the side of the block facing the obstacle. If the profile has a different shape, the smallest rectangle or circle that will contain the profile is used.

2.2.3. Safe_z_plane

The milling area model also includes a “safe_z_plane”. This is a horizontal plane above which there are no side obstacles.

Figure 3. Milling Area Model



2.2.4. Automatic Updating of Vise Obstacle Data

The coordinate system in which side and bottom obstacles are represented in the Data Execution module is the coordinate system of the part. To be independent of the part, however, the world model keeps obstacle data in the coordinate system of the vise. Since the part is always placed in the center of the vise, it is feasible to update the world model to represent obstacles accurately for parts of differing sizes placed in the vise.

Figure 4 is a top view of the vise with a workpiece in it. As shown in Figure 4, the bottom obstacles presented by the vise include the two parallel jaws of the vise (parallel to the x-axis) and the two crossbars of the vise (parallel to the y-axis). The crossbars slide through holes in the upper (fixed) jaw of the vise so that their effective length changes according to the width of the workpiece.

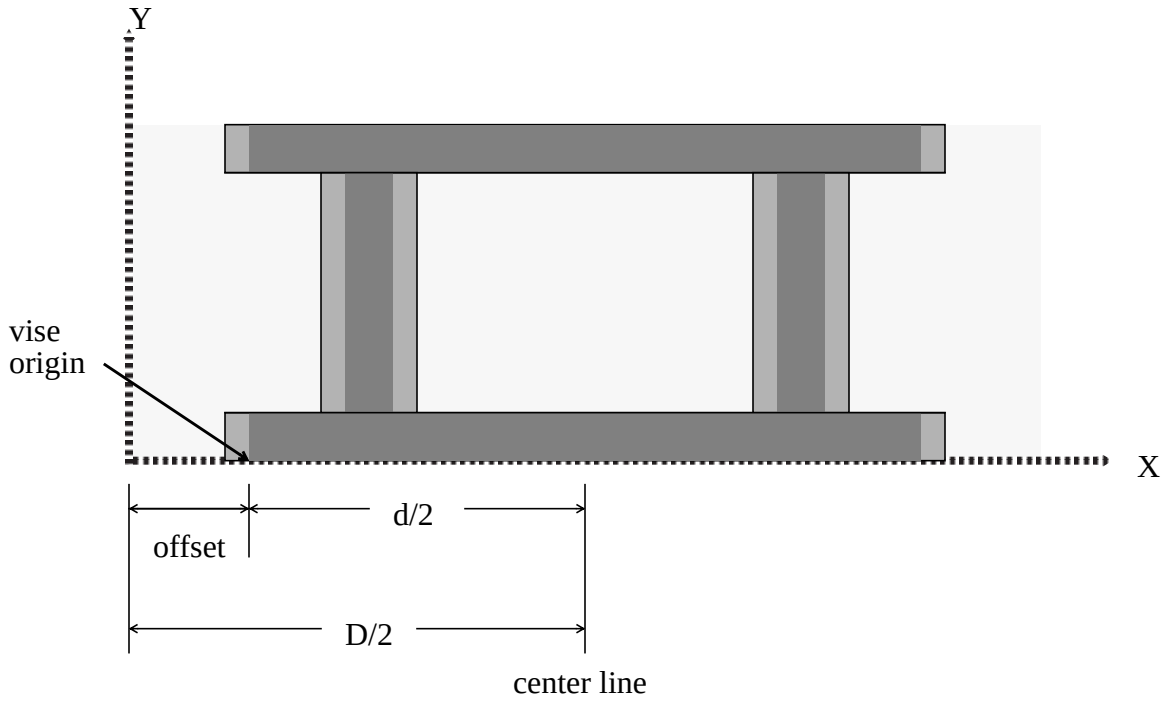
In the database, the vise bottom obstacles are represented with the origin of the coordinate system at the lower left corner of the workpiece, as shown in Figure 4. To change the obstacles to workpiece coordinates the x-values and the y-values of the obstacles must both be changed.

All x-values for the obstacles are changed by adding the offset amount. This is half the difference between the length of the workpiece and the length of the vise. This adjustment is correct regardless of which is greater.

The y-values for the lower vise jaw and the lower right corners of the crossbar obstacles do not change. The y-value for the upper left corner of the upper jaw is equal to the width of the workpiece. The y-values for the lower right corner of the upper jaw and the upper left corners of the crossbar obstacles are equal to the width of the workpiece minus the width of the jaw.

Although they are not shown in Figure 4, the x-values of the side obstacles presented by the vise are updated the same as the x-values of the bottom obstacles.

Figure 4 . Updating Vise Obstacles



This is a top view of a rectangular workpiece sitting in the vise. The sides of the vise which clamp the workpiece in place are not shown.

The workpiece is shown in very light gray. The length of the workpiece is D.

The part of the vise immediately under the workpiece is shown in solid gray. Because the workpiece may not be centered exactly in the vise, the extent of the vise in the x direction is increased by a quarter inch, shown in lighter gray. The length of the vise is d.

The coordinate system of the part is shown in heavy dotted lines.

The bottom obstacles for the workpiece are the four rectangles outlined with light solid lines.

The offset used to adjust x-values of bottom obstacles is (D/2 - d/2).

2.2.5. Machining Parameters

Two machining parameters are used which are not part of the milling area model but are closely related to it: max_thru and min_thick. Max_thru is the maximum amount that a tool is to be allowed to go past the bottom of the part. Max_thru is set to be equal to or less than the bottom_clearance. Min_thick is the minimum thickness that may be left between certain features and the outside of the part. For example, if a pocket is to be made, the bottom of the pocket must be at least min_thick thick or the pocket must go all the way through the part.

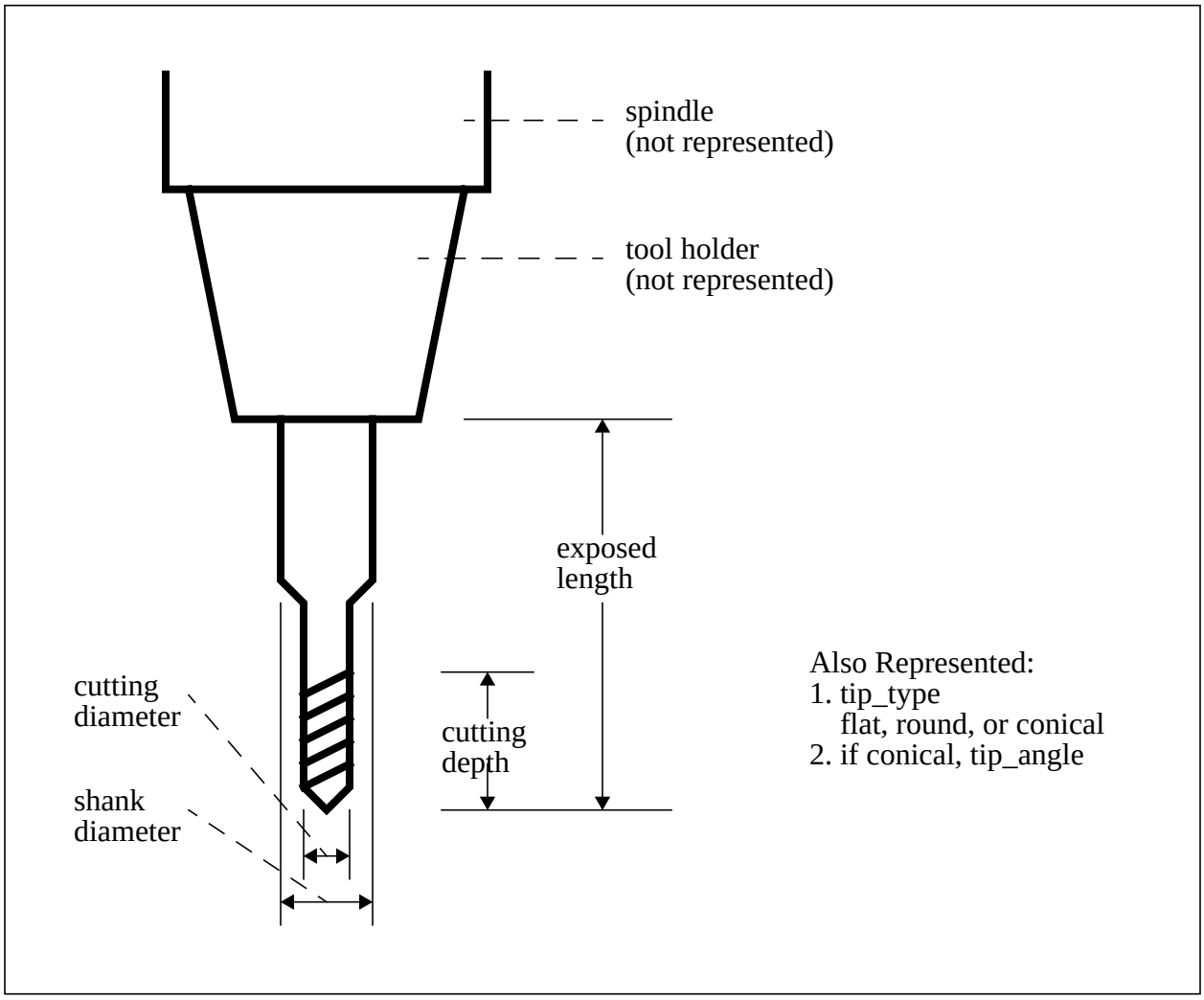
2.3. Tool Geometry Model

The tool geometry model implicit in the system is shown in Figure 5. The model includes a tool and its holder. Tool parameters include cutting depth, cutting diameter, shank diameter and exposed length. "Cutting depth" is the vertical amount of the tool that has flutes on it for cutting into the part. "Cutting diameter" is the diameter of the widest part of the tool within the cutting depth. "Shank diameter" is the diameter of the widest portion of the tool above where the flutes stop and below the base of the holder. The "exposed length" is the length from the tip of the tool to the base of the tool holder. The values of these parameters, and other, more specific information which varies with the tool type, are kept in world model.

The dimensions of the tool holder and the spindle of the milling machine are not represented in the tool model because at no time is the base of the tool holder allowed below the surface of the part.

Each operation verifier gets the description of the tool to be used during the specific operation by extracting the description of the tool in the slot specified by the process plan from the world model.

Figure 5. Tool Geometry Model



3. CHECKS INCLUDED IN OPERATION VERIFIERS

3.1. Introduction

Most of the operation verifiers make the following ten checks. Not all operation verifiers perform all ten tests, but many have additional tests listed in section 3.12.

3.2. Tool Must Be Suitable

The tool selected for an operation must be suitable to carry it out. The tests vary with feature type. Examples: (1) grooves dictate the size of the tool by how wide the feature is, (2) for pockets, the radius of the tool must be smaller than the corner radius of the feature, (3) the tool used to make chamfers should always have a 90 degree tip angle. In some cases there is a minimum tool size.

3.2.1. Center_drill

The tool must be a center drill at least 0.125 inch in diameter. The feature must be a hole.

3.2.2. Counterbore

The tool must be an end_mill the same diameter as the feature being counterbored. The diameter must be at least 0.125 inch. If the feature is a hole, it must not have a conical bottom.

3.2.3. Drill_hole

The tool must be a drill the same diameter as the feature being drilled. The diameter of the drill must not be less than 0.0624 inch. The diameter of the shank of the drill must be the same as the diameter of the cutting portion of the drill.

3.2.4. Face_mill

The tool must be a face_mill at least 0.75 inch in diameter.

3.2.5. Fly_cut

The tool must be a fly_cutter at least 1.0 inch in diameter.

3.2.6. Machine_chamfer_in

The tool must be a chamfer with a tip angle of 90 degrees. The radius of the feature being chamfered must not be less than half the radius of the chamfer tool (since chamfering is done with the side of the feature being chamfered meeting the tool at half its cutting depth). The tool must be at least 0.125 inch in diameter. The feature being chamfered must be a pocket, groove, straight_groove, or hole.

3.2.7. Machine_chamfer_out

Recall that a chamfer_out may be a feature (in which case the outer edge of the block gets chamfered) or a subfeature of a groove (in which case the island left inside the groove gets chamfered).

The tool must be a chamfer with a tip angle of 90 degrees. The tool must be at least 0.125 inch in diameter. If the chamfer is a subfeature, the parent feature must be a groove.

3.2.8. Machine_countersink

The tool must be a countersink at least 0.125 inch in diameter. The feature being countersunk must be a hole. If there is a reference feature, its bottom must not be conical.

3.2.9. Mill_contour_groove

The tool must be an end_mill or ball_nosed_end_mill of the size required to make the groove. The tool diameter must be at least 0.125 inches.

3.2.10. Mill_contour_pocket

The tool must be an end_mill at least 0.125 inches in diameter whose radius is at least 0.01 inch smaller than the radius of the tightest corner into which the tool must fit.

3.2.11. Mill_groove

The tool must be an end_mill or ball_nosed_end_mill of the size required to make the groove. The tool diameter must be at least 0.125 inches. The radius of the tool must not be larger than the corner radius of the groove.

3.2.12. Mill_pocket

The tool must be an end_mill at least 0.125 inches in diameter. It must fit within the pocket and its radius must not be greater than the corner radius of the pocket.

3.2.13. Mill_side_contour

The tool must be an end_mill at least 0.125 inches in diameter whose radius is at least 0.01 inch smaller than the radius of the tightest corner into which the tool must fit (if there are any concave corners).

3.2.14. Mill_straight_groove

The tool must be an end_mill or ball_nosed_end_mill of the size required to make the groove. The tool diameter must be at least 0.125 inches.

3.2.15. Mill_text

The tool must be a ball_nosed_end_mill of the size required to make the text. The tool diameter must be at least 0.125 inches.

3.2.16. Set0_center, Set0_corner, and Set0_z

The tool must be a probe.

3.2.17. Tap_thread

The tool must be a tap whose cutting diameter is the same as the thread diameter of the hole being tapped. The number of threads per inch of the tool must be the same as the number of threads per inch of the hole. The diameter of the tool must not be less than 0.0624 inch.

3.3. Tool Holder Must Stay Above Top of Part

Since the spindle and the tool holder are not explicitly represented in the world model, the total depth of a cut cannot exceed the exposed length of the tool. This prevents the tool holder base from ever going below the top surface of the part. This check is applied to every operation except: face_mill (see section 3.12.2), fly_cut (see section 3.12.2), and the three zero_setting operations.

3.4. Shank Must Clear Reference Features

If a tool is inserted deeper than the cutting depth of the tool below the surface of the part, then the shank of the tool becomes a consideration. If the shank's diameter is larger than the cutting diameter, a check is made that the shank does not strike any reference feature in the nest of reference features above the feature being cut. This calculation requires the use of "virtual" features, as described in subsection 3.6.

This check is applied for the following operations: center_drill, counterbore, machine_chamfer_in, machine_chamfer_out, machine_countersink, mill_contour_groove, mill_contour_pocket, mill_groove, mill_pocket, mill_side_contour, mill_straight_groove, and tap_thread.

This check is not applied for: drill_hole, face_mill (see section 3.12.2), fly_cut (see section 3.12.2), and the three zero_setting operations.

3.5. Do Not Cut Deeper than Flutes

While milling a part, the tool should never be required to cut deeper than its cutting depth. For a drill, this is necessary to allow chips to be removed properly. For tools which cut on their sides, it is unwise to try to cut with the shank of the tool. This check is applied for every operation except: face_mill, fly_cut, and the three zero_setting operations.

In the case of machine_chamfer_in and machine_chamfer_out, the limit is half the cutting depth, since only the upper half of the tool is used.

In the case of face_mill and fly_cut, since all the material on the top of the part is removed on each pass, it is feasible to cut deeper than the cutting depth of the tool, as long as it is done in several passes.

3.6. Tool Tip Must Clear Workpiece

When several features are nested or if a narrow feature is being cut, there exist possibilities for damaging features on higher levels or attempting to insert a tool in a space that is too small. This may be the case if the tool tip is conical (i.e. chamfer, countersink, center drill) or ball nosed. A chamfer tool also might hit the bottom of the feature. Figure 6 shows two tools inside a nest of three reference features striking the reference features.

This is not a problem for end mills since the tool is cylindrical and must fit within the feature being milled.

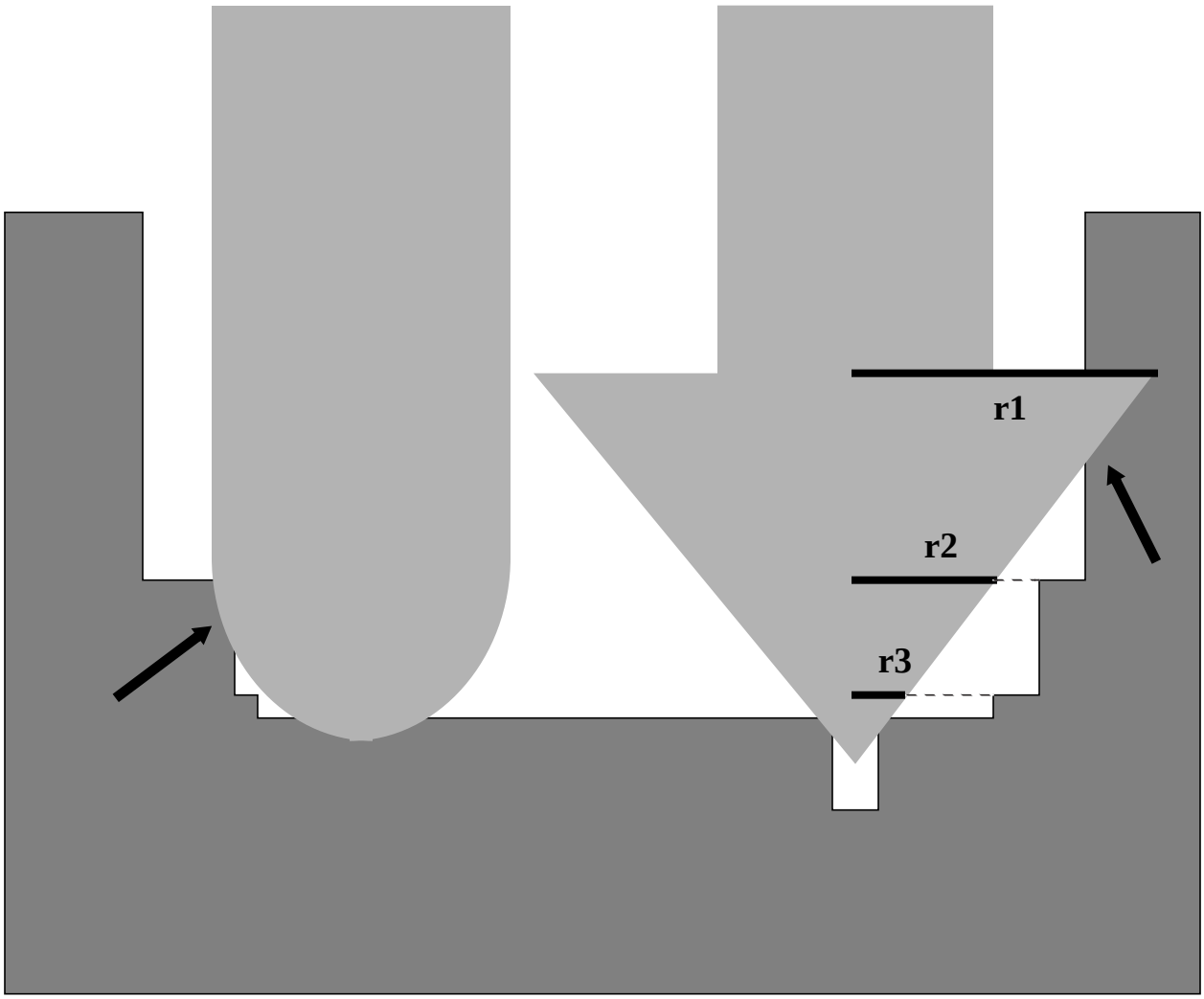
This check is applied for the following operations: center_drill, machine_chamfer_in, machine_chamfer_out, machine_countersink, mill_contour_groove (if the groove is round bottomed), mill_groove (if the groove is round bottomed), mill_straight_groove (if the groove is round bottomed), and mill_text.

This check is not applied for: counterbore, drill_hole, face_mill, fly_cut, mill_contour_pocket, mill_pocket, mill_side_contour, the three zero_setting operations, and tap_thread.

Calculating whether the side of the tip of the tool will hit a reference feature involves the use of “virtual” features. A virtual feature is constructed by finding the path the center of the tool must follow to make the outermost extent of the feature or subfeature being machined and using a “virtual radius” for the tool. The outline of the virtual feature is one virtual radius outside of the path of the center of the tool. A separate virtual feature is constructed for each reference feature in a nest of reference features which might be hit by the tip of the tool. A check is then made of whether the outline of each virtual feature lies within the outline of the corresponding reference feature. These checks of virtual features against several reference features may require a great deal of calculation.

In Figure 6 three virtual radii are shown on the conical tool. The uppermost of the three, r1, is the maximum radius of the tip. The other two virtual radii, r2 and r3, are the radii of the tool at the height of the top of the corresponding reference feature, where the tool is most likely to hit the feature.

Figure 6. Tools Hitting Reference Feature



Two tools are shown in this side view, making cuts in the bottom of a nest of three pockets inside a part. On the left, a ball-nosed tool is making a groove. On the right a tool with a conical tip is countersinking a small hole in the bottom of the lowest pocket. Both tools are making the correct cuts, but they are striking reference features at the places shown by the heavy arrows. This kind of situation is detected automatically by the verification subsystem.

The three virtual radii needed by the verification subsystem to check interferences of the conical tool with the three reference features are shown with solid lines as r1, r2 and r3.

3.7. Stay Within Part or Machining Box

For some feature types, the tool envelope should stay within the outline of the block. In any event, since the world outside the machining box is unknown, the tool envelope should stay inside the machining box. In some cases the system checks that the tool envelope is inside both the part and the bounding box. This is so that if the user elects to proceed in spite of one error, the user will still be notified in case of a second error.

In the three zero_setting operations, the only requirement is that the location of the probe in the xy-plane (which is required by all three) is within the rectangle where $15 < x < 40$ and $0 < y < 12$. This is the part of the table of the milling machine where workpieces are fixtured. It includes both the vise and the pallet clamping area.

3.7.1. On the Sides

Keep the tool within the sides of the part by at least the amount min_thick for: center_drill, counterbore, drill_hole, machine_countersink, mill_contour_groove, mill_text, and tap_thread. In the cases of center_drill and machine_countersink, the portion of the tool above the top surface of the part may go beyond the “min_thick” limit.

If the tool does not go through a given side of the part, it must stay at least min_thick within that side for: mill_contour_pocket, mill_groove, mill_pocket, and mill_straight_groove.

Stay within the bounding box for: face_mill, fly_cut, machine_chamfer_in, machine_chamfer_out, mill_contour_groove, mill_contour_pocket, mill_groove, mill_pocket, mill_side_contour, mill_straight_groove, and mill_text.

In the cases of face_mill, fly_cut, machine_chamfer_in, and machine_chamfer_out, knowledge of the cutting algorithm is required to find the amounts the tool will run off the part in the four directions.

3.7.2. On the Bottom

Do not go more than max_thru past the bottom of the part for: center_drill, counterbore, drill_hole, machine_chamfer_in, machine_countersink, mill_contour_pocket, mill_pocket, mill_straight_groove, and tap_thread.

The feature may not go thru the part, so leave the bottom at least min_thick thick for: mill_contour_groove, mill_groove, mill_side_contour, and mill_text.

If the feature is not a thru feature, leave the bottom at least min_thick thick for: counterbore, mill_contour_pocket, mill_pocket, and mill_straight_groove.

Leave at least min_thick left of the part for: face_mill and fly_cut.

In the case of machine_chamfer_out, if the block is being chamfered, the tip of the tool must not go below the safe_z_plane, and if a groove is being chamfered, since the tip of the tool may not go below the bottom of the feature (through grooves are not allowed), no other check is made at the bottom.

3.8. Do Not Hit Obstacles

If a tool passes through a side or the bottom of the block, be sure it does not hit any obstacles.

The three zero-setting operations do not worry about any obstacles.

3.8.1. Side Obstacles

A check is made of side obstacles for: machine_chamfer_in, machine_chamfer_out, mill_groove, mill_pocket, and mill_straight_groove.

For center_drill and machine_countersink, the tool may not go outside the part below the safe_z_plane, so no check of side obstacles is necessary.

Since the tool or the feature is required to stay within the sides of the part, no check is made of side obstacles for: counterbore, drill_hole, mill_contour_groove, mill_contour_pocket, mill_text, and tap_thread. This is a minor loophole in the system as follows. If a pocket is made through a side of the block, then the part is refixtured with a side obstacle against the open side of the pocket, and one of these operations is carried out at the bottom of the pocket so that the cutting part of the tool stays inside the part by min_thick, the tool could hit the side obstacle even though no error message is generated. If the pocket is made with the same fixturing as the other operations, the others would be OK if the pocket-making operation was OK, since the tool for the other operations must stay within the pocket.

For face_mill, fly_cut, and mill_side_contour, the tool is required to stay above the safe_z_plane. Since the safe_z_plane is above all side obstacles, no check of side_obstacles is needed.

3.8.2. Bottom Obstacles

A check is made of bottom obstacles if the tool cuts through the bottom of the part for: center_drill, counterbore, drill_hole, machine_chamfer_in, machine_countersink, mill_contour_pocket, mill_pocket, mill_straight_groove, and tap_thread.

For conical or ball-nosed tools passing through the bottom of the workpiece, the actual diameter of the hole in the bottom of the workpiece (which may be less than the diameter of the tool if only the tip of the tool pokes through) is used for checking against bottom obstacles.

3.9. Spindle Speed Must Be in Reasonable Range

If the speed is within the range from one half the system calculated value to twice it (not to exceed 5200 revolutions per minute -- which is the maximum speed the machine will produce), then the test is passed. The system uses the tool type, its diameter and the material to calculate a reasonable speed. This test applies to all operations except the three zero-setting operations.

3.10. Feed Rate Must Not Be Too High

There is no lower limit, but the upper limit is twice the system calculated feed rate, not to exceed 30 inches per minute. The system calculates feed rate with respect to the speed, material, tool type and cutting diameter. This test applies to all operations except the three zero-setting operations and tap_thread.

In the case of tap_thread, the tapping cycle requires the fake value of 300 for the feed rate, and the verification subsystem checks for the value 300. The tap does not actually feed at this rate. Rather, the spindle speed is set, the vertical motion of the spindle is set free, and the feed rate is determined by how fast the tap screws itself into the workpiece.

3.11. Pass Depth Must Be in Reasonable Range

Pass depth is an incremental depth used to cut deep features in several passes. The system calculates pass depth according to the material and the tool used to make the feature. Attempting to cut too deeply can damage the tool or cause tolerance errors in the part. A maximum of ten passes is allowed. Not all operations use a pass depth. The pass_depth in the process plan must not be more than 1.5 times the pass_depth calculated by the system.

A pass depth check is made for: drill_hole, face_mill, fly_cut, mill_contour_groove, mill_contour_pocket, mill_groove, mill_pocket, mill_side_contour, mill_straight_groove, and mill_text.

A pass depth check is not made for: center_drill, counterbore, machine_chamfer_in, machine_chamfer_out, machine_countersink, and tap_thread -- all of which are done in one pass.

A pass depth check is not made for the three zero-setting operations.

3.12. Additional Tests

3.12.1. Center_drill

If there is a reference feature, it must be flat bottomed. In other words, do not center drill in a conical bottomed hole.

The center drill must not go deeper than the bottom of the hole if the hole is a blind hole, and must not cut material the twist drill will not cut in the case of a partially thru hole.

3.12.2. Face_mill and Fly_cut

In the cases of face_mill and fly_cut, the cutter must stay above the safe_z_plane, which is assumed to keep the holder clear of the part. Since face_mill and fly_cut do not make features and are assumed to be applied at the top of the part, no check of hitting reference features is required.

3.12.3. Mill_contour_pocket

The tool must not mill into the far side of the contour outline while cutting the near side. For example, imagine an hourglass shaped contour_pocket; the tool must be small enough to cut one side of the neck of the hourglass without cutting into the other side of the neck. The radial stepover used for the cut must be in the range from half to twice the value calculated by the system, not to exceed the tool diameter minus 0.01 inch.

3.12.4. Mill_pocket

The radial stepover used for the cut must be in the range from half to twice the value calculated by the system, not to exceed the tool diameter minus 0.01 inch.

3.12.5. Mill_side_contour

As with mill_contour_pocket, the tool must not mill into the far side of the contour outline while cutting the near side. In addition, a check is made that the contour itself fits inside the part. This is because the algorithm for generating NC-code to cut a side_contour will not work if the contour goes much outside the part. The radial stepover used for the cut must be in the range from half to twice the value calculated by the system, not to exceed the tool diameter minus 0.01 inch.

3.12.6. Set0_corner

The corner number must be 1, 2, 3, or 4 (representing the four corner types of a rectangle, starting with 1 at the lower left, and proceeding counterclockwise around the rectangle). The sides of the corner are assumed to be straight and parallel to either the x-axis or the y-axis.

3.12.7. Tap_thread

If a hole is being tapped that is not a clean thru hole, an allowance for chips of three quarters of the tap diameter must be left at the bottom of the hole.

4. SOFTWARE FOR MACHINING OPERATION VERIFICATION

The functions which perform the verification of machining operations are called by the Data Execution module. That module selects one of 21 tests from the “machine_ops” data base, according to the operation to be executed (there are 21 operations). The 21 test functions all are in the “verify2” subdirectory of the vws2 directory, and they all end in the suffix “test”. Another 19 functions in the same directory are subordinate only to the top 21. In addition, however, the entire machinery of the geometry library and the reference feature fit checking functions are necessary to complete the work of the operation verifiers. These other functions are required to generate and test virtual features for making sure the shank and tip of the tool do not hit the workpiece.

In addition, the functions in the proc2 and exec2 subdirectories of the vws2 directory which calculate spindle speeds, feed rates, pass depth and stepover are called by the machining operation verifiers.

VII. **WORKPIECE VERIFICATION**

A workpiece in the VWS2 system is represented by a data structure very much like the data structure for a design. It consists of a header and a list of features. A drawing of the original workpiece will be made by the Data Execution module before it starts through the process plan, if the drawing option is on. A workpiece drawing may also be obtained directly through the vws_cadm user interface. In either case, if the verification mode is on_soft or on_hard, the workpiece will be verified as it is drawn. The verification of a workpiece is done by the same functions that verify a design.

VIII. **PART MODEL CHECKING**

When the Data Execution module works its way through a process plan, the only thing it always does is to build a data model of the workpiece. All other actions are optional.

The data model starts out as a model of the incoming workpiece, which may be a featureless block or may contain some of the features in the design which is being used by the process plan. As a step in the plan is executed, the feature or subfeature made by the step is added to the workpiece model.

Before a feature or subfeature is added to the model, checks are made as follows (regardless of the setting the verification mode): 1. The feature or subfeature must not already exist. 2. If there is a reference feature, it must already have been made. 3. For a subfeature or a counterbore, the parent feature must already have been made.

IX. **OTHER AUTOMATIC VERIFICATION**

1. GENERAL

In addition to the verifiers described in other sections of this paper, dozens of checks are scattered throughout the VWS2 system. There is no clear dividing line between checks specific to automated machining and ordinary good programming practice checks, such as data type checks. There are many instances of both. For example:

A. The Design Editor checks whether a file already exists for a design of the given name when the user asks to have a design saved. If it does, the user is asked whether it is OK to write over the old file.
B. The Process Planning module checks that the reference features in a design have been assigned sensibly.
C. The Data Execution module checks that all the tools called for in a process plan are present on the milling machine before it starts executing a process plan.
D. The vws_cadm user interface checks whether a data structure already exists of a given name when the user selects a name for a process plan or other data structure.

2. INITIATION OF THE DATA EXECUTION MODULE

2.1. Introduction

The Data Execution module makes a number of checks when it initiates execution of a process plan. The checks are made by the design enhancement and design verification subsystems, plus the functions enhance1_plan, execute_plan, init_exec_plan, init_fixture, precify, subdesign, and update_vise_obs. If any of these checks fails, a specific message is sent to the user.

2.2. Design Checks

A. The design is enhanced and checked during enhancement as described in Chapter IV.
B. If verification is on, the design is verified as described in Chapter V.

2.3. Process Plan Checks

A. The steps in a plan must be numbered sequentially starting with 1.
B. Step 1 must be initialize_plan.
C. The last step must be close_plan.
D. There must be no loops of precedent steps embedded in the assignment of precedent steps.
E. The assignment of precedent steps must force close_plan to be executed last.

2.4. Workpiece Checks

A. The workpiece length and width must be within 0.00001 inch of the design length and width.
B. If the workpiece has no features, its height must not be less than the design height or more than an inch greater than the design height.
C. If there is a slab on top of the workpiece, the height of the workpiece after removing the slab must not be less than the design height.
D. If there are features on the workpiece, they must be the same as features in the design, except that subfeatures may be lacking.
E. The design_id listed in the header of the workpiece description must be the same as the design_id of the design.
F. If the drawing and verification options are on, the workpiece is verified as described in Chapter VII.

2.5. Fixturing Checks and Machining Checks

A. The value of the location from which to locate z-zero must be “fixture” or “top_of_part”.
B. The name of the fixture must be “pallet” or “vise”.
C. All the tools required by the process plan must be present on the machine.
D. If the part is to be fixtured in the vise, it must not be too narrow or too wide for the minimum and maximum opening of the vise.
E. If the part is to be fixtured in the vise, it must not be too long.

X. **DESIGN DRAWING**

Design drawing is the first of the three types of user-interactive verification. All three are graphical. In each case the system draws a picture and the user examines the picture and uses his or her judgement to decide if what is shown is the way things should be.

The picture of a design (or a workpiece model) is a three-view mechanical drawing lacking some features commonly available on commercial systems, such as dimensioning.

In design drawing verification the system draws a picture of the part as it is being designed and the user decides if the design shown is what was intended. This kind of verification is very important, but so obvious it is often not recognized as being verification. Until computers can make good guesses about what a human intends, this kind of verification cannot be automated.

Design drawing is also available directly from the vws_cadm user interface. The design cannot be edited in this mode, but the user can make judgements about the correctness of the design.

XI. **WORKPIECE MODEL DRAWING**

Workpiece model drawing is available only in the Data Execution module. That module maintains an internal data structure representing a model of a workpiece. As the module goes through a process plan, the data model is updated after each step of the plan is executed. If the user asks for it, the module will draw a picture of the model as it is updated. This provides a graphical simulation of the machining process. By watching the picture change, the user can judge if the order of machining operations is suitable.

XII. **TOOL PATH DRAWING**

Tool path drawing is, as the name implies, a drawing of the path taken by the tool when an NC program is executed. The user may examine the path to see if the cuts appear to be made properly.

Tool path drawing is available from the vws_cadm user interface only immediately after the Data Execution module has generated an NC code file with the drawing option on. That module actually writes pseudocode (in a format easy for LISP to handle) when it executes a process plan. The real NC code is generated at the end of the module's operation by feeding the pseudocode through a printing routine. The tool path drawing is made from the pseudocode, not from the real code. The pseudocode is discarded after the real code is printed. That is why tool path drawing is available only in these circumstances.

At the user's option, the path will be drawn slowly (it is usually drawn too quickly to follow). A second option is to have the drawing stop each time the nc-code switches from contour cutting to traversing or a spindle retract. In this option the user gives a command to continue the drawing.

The tool path drawing is a line on all three views of the workpiece that follows the point at the center of the tip of the tool. An example is shown in Figure 7. When the tool is moving at traverse rate (and should not be cutting), a heavy dotted line is drawn. When the tool is not moving at traverse rate (and is probably cutting), the line is thin. Thin lines are dotted on the top view and solid on the front and side views in order that they may be seen more clearly.

On the front and side views, vertical motions are cut off one inch above the workpiece so that the top view is not obscured. Spindle retracts normally lift the tool higher than one inch above the workpiece.

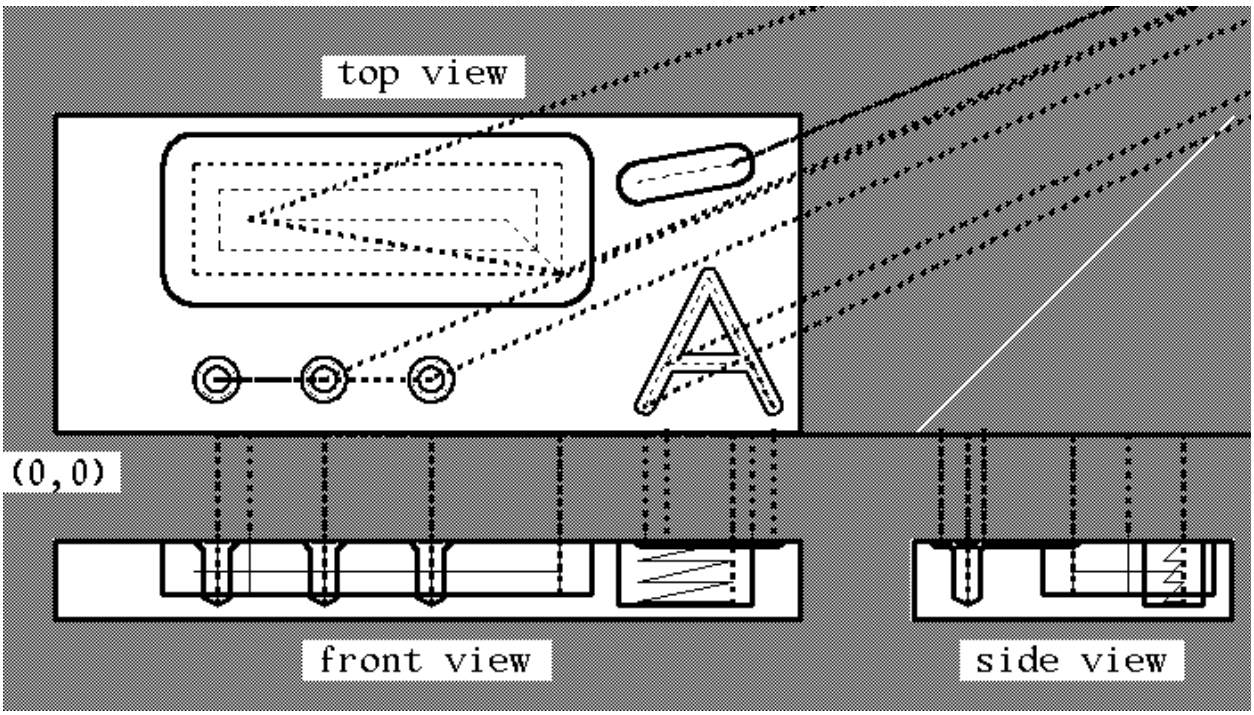
In Figure 7 thin lines may be seen behind dotted lines on the side views of all the holes. The heavy lines going toward the upper right of Figure 7 show where the tool is moved away from the workpiece to be changed.

In principle, we believe that having a human examine a tool path drawing is an outmoded form of verification. If (1) the process plan has been verified, (2) the cutting algorithms embody sound machining practice, and (3) the NC code generating routines implement the cutting algorithms correctly, then it should be safe to assume that the NC code is correct without further checking.

An analogy (due to our colleague J. Jun) may be made with compilers for computer languages. Two decades ago programmers examined the output of compilers to make sure the assembly language code was correct. Now programmers are confident enough of the correctness of compilers that assembly language code is rarely examined. We believe the same thing will happen in the operation of NC machines. If the process plan is correct, it will be safe to assume that the output of NC code generators is correct. That point has not yet been reached, but we believe that research efforts directed to improving tool path drawings might better be applied elsewhere.

The tool path drawing facility, in fact, was included in the system not for checking NC code, but for debugging the NC code generator.

Figure 7. Tool Path Drawing



The picture shows the tool path required to make a pocket, a text feature, three holes, and a straight_groove in a workpiece. The dotted lines and the thin lines represent the path of the center point of the tool. Heavy dotted lines show when the tool is moving at traverse rate and should not be cutting the workpiece. Thin dotted and solid lines show when the tool is moving slowly and is probably cutting.

On the front and side views, vertical motions of the tool are cut off one inch above the workpiece so that the top view is not obscured.

The heavy diagonal lines show where the tool is moved to a safe place away from the workpiece to be changed.

XIII. AUTOMATIC GENERATION OF FEATURE VERIFIERS

1. INTRODUCTION

As noted earlier, the feature verifiers in the VWS2 system were written by a rule-based automatic programming system. The feature verifiers are LISP functions, one for each feature type, which check if a feature of the given type conforms to a set of rules.

The user of the automatic programming system writes rules in English and the system generates LISP code to get the rules applied. The allowable syntax for writing rules is very tightly constrained, but it is English. Only words and phrases determined beforehand are allowed to be used.

Here are four examples of rules that are part of the VWS2 system.

A. The y_value of the plus_y side of the groove minus the y_value of the minus_y side of the groove should be greater than 2.0 times the width of the groove plus the chamfer_in_depth of the groove plus the chamfer_out_depth of the groove.

B. The set of text characters should be contained in the set of all machinable characters.

C. The chamfer_in_depth of the pocket should not be greater than the vertical rise of the pocket.

D. If the chamfer_in_depth of the groove is greater than zero, then the x1 of the groove should be close to the x2 of the groove, or the y1 of the groove should be close to the y2 of the groove.

Some of the rules, such as B and C above, are included because the feature description would not make sense if the rule were violated. Some are included to avoid undesirable design conditions, such as A above, where a violation would result in overlapping chamfers. Some, such as D above, which requires that straight_grooves be horizontal or vertical if they are to be chamfered, are included because the NC-coding capability of the system cannot handle a particular situation. Some, such as not allowing a pocket to cut a part in two, are included to avoid dangerous machining conditions. And, finally, some are included because the drawing subsystem will not draw the feature correctly if the rule is violated.

An example of a feature verifier written by the automatic programming system is shown in Figure 8, divided into sections.

Figure 8. Feature Verification Function

```
(def new_verify_hole
  (lambda (desc block_size)
    (let ((ok_flag t)
          root_tpi
          (threads_per_inch (get desc 'threads_per_inch))
          (countersink_diameter (or (get desc 'countersink_diameter) 0.0))
          (diameter (get desc 'diameter)))
        (setq root_tpi (sqrt threads_per_inch))
        (cond ((not (lessp root_tpi 5.0))
               (setq ok_flag nil)
               (erer_mesij '("The square root of the number of threads per inch"
                             "should be less than 5.0.")))))
        (cond ((not (greaterp (quotient countersink_diameter diameter) 1.05))
               (setq ok_flag nil)
               (erer_mesij '("The countersink_diameter of the hole divided by the"
                             "diameter of the hole should be greater than 1.05."))))
        ok_flag)))
```

A

B

C

D

E

F

2. HOW THE AUTOMATIC PROGRAMMING SYSTEM WORKS

The automatic programming system does not parse the rules into parts of speech or types of phrases. It simply converts each rule into a LISP "cond" function through a series of transformations and embeds the "cond" functions in a LISP function with other items needed to make the "cond" functions work. If any rule is violated, the rule is printed out for the user to see. To make them available for printing, the rules are included in the function in readily printable form.

A feature verification function has the following parts:
A. two lines of LISP code that begin function definitions (section A of Figure 8),
B. a "let" LISP statement to establish local variables and set the value of some of them (section B of Figure 8),
C. a "setq" LISP statement to give values to any variables that didn't get a value assigned in the preceding section (section C of Figure 8),
D. a series of "cond" functions, one for each rule, (sections D and E of Figure 8), and
E. one closing line of code to return "t" if no rule is violated or "nil" if any rule is violated.

Many of the transformation algorithms for converting English text into LISP code are hard-coded. This includes all the algorithms for dealing with arithmetic statements. Other information needed to transform English into LISP is stored in the "features" data base. For each feature type, this includes a set of local variables to use, a directory that tells what local variables to substitute for selected English phrases, and a directory of how to calculate other local variables.

3. FORMAT FOR RULES

The range of acceptable formats for rules is shown in Table 3, defined in BNF. Although it is not obvious from the table (BNF definitions tend to be hard to read), the BNF definition requires a period at the end of a rule. Each feature type has its own list of feature noun phrases that are allowable for inclusion in a rule about that feature type. As an example, the feature noun phrases acceptable to use with holes are included with the BNF definition in Table 3.

In addition to requiring that rules satisfy the BNF definition, the following requirements and conventions are observed:
A. The file containing the rules must have a blank line between rules.
B. The file containing the rules must end with the word "end" preceded by a blank line.
C. In a mixture of "and's" and "or's", the "or's" are superior in the sense that
"A and B or C and D" means "(A and B) or (C and D)" rather than
"A and (B or C) and D"
D. The BNF definition allows the use of parentheses in arithmetic statements. If a series of arithmetic operations is given without parentheses, however, it is assumed to be intended to be parenthesized according to the following model:
"A plus B minus C times D" means "(A plus (B minus (C times D)))".

E. Each rule must make sense. For example if arithmetic operations are to be performed, they must be performed on numbers.
F. The first letter of a rule may always be capitalized (it should be in order to be good English, but the system does not check).
G. Commas may be used in rules. The system ignores them. It is intended that commas be used where they are required for correct English.

Table 3
BNF Definition of an Acceptable Verification Rule

<rule> ::= <assertions>. | If <is_assertion> {and|or <is_assertion>} then <assertions>.

<assertions> ::= <should_assertion> {and|or <should_assertion>}
| It|it should [not] be that <is_assertion> {and|or <is_assertion>}

<is_assertion> ::= <computed_phrase> is [not] <comparative> <computed_phrase>

<should_assertion> ::= <computed_phrase> should [not] be <comparative>
<computed_phrase>

<computed_phrase> ::= <noun_phrase> {<arithmetic_verb> <noun_phrase>}
| (<computed_phrase>)

<comparative> ::= greater than | less than | equal to | contained in | close to

<arithmetic_verb> ::= plus | minus | times | divided by

<noun_phrase> ::= zero | nil | <number> *i.e. a real number*
| <machining_parameter> | <block_noun_phrase> | <feature_noun_phrase>

<machining_parameter> ::= min_thick

<block_noun_phrase> ::= The|the <block_term> of the block

<block_term> ::= x_value of the plus_x side | x_value of the minus_x side
| y_value of the plus_y side | y_value of the minus_y side
|z_value of the top | z_value of the bottom | length | width | height

Acceptable values for <feature_noun_phrase> depend upon the feature.
The values acceptable for holes are given here as an example.
*These phrases all follow the format "The **** of the hole".*
Most feature noun phrases in the system follow this format, but some do not.

<feature_noun_phrase> ::= The|the <feature_term> of the hole

<feature_term> ::= x_value of the center | y_value of the center | diameter
| radius | chamfer_in_depth | threads_per_inch | thread_diameter
| thread_depth | given depth | total depth | vertical rise | bottom_type
| x_value of the plus_x side | x_value of the minus_x side
| y_value of the plus_y side | y_value of the minus_y side
| z_value of the top | z_value of the bottom | countersink_diameter

4. AN EXAMPLE

To make a verification function for a feature type, the user writes a file of rules in acceptable format, using only feature noun phrases appropriate to that feature type. The user then calls the "define_verifier" function, the arguments to which are (i) the feature type, (ii) the name of the file where the rules are given, and (iii) the name the user wants to give to the function that will be defined. The system defines the function, taking a few seconds to a few minutes to do so. Then the user may use the LISP pretty-printer to write the function to a file in decent format. An expert user can expand the list of phrases the system will accept by editing the features data base before calling define_verifier.

Suppose a user wanted to change the rules for a hole so that only the two following rules were applied: (1) The square root of the number of threads per inch should be less than 5, and (2) The ratio of the countersink diameter of the hole to its diameter should be more than 1.05.

For rule 1, a calculation function not known to the system is required, so the user would have to invent a term for "the square root of the number of threads per inch" -- let's use "root_tpi". The user would make the following entries in the part of the file ~/vws2/datab2/world/verify_data.l that deals with holes to tell the system how to deal with the new term:

A. In the section establishing "let" variables, add the term "root_tpi" at the end.
B. In the "setq" section, which defines how values are assigned, add the line:
   "root_tpi (sqrt threads_per_inch)".
C. In the section defining the words or phrases equivalent to variables, which is the
   "rule_frags" section, add the line:
   "root_tpi ((the square root of the number of threads per inch))".

For rule 2, the system recognizes "divided by" but not "ratio", recognizes "greater than" but not "more than", and recognizes "the diameter of the hole" but not "its diameter". However, there are no items or calculations unknown to the system, so we need only rephrase rule 2 as follows: The countersink_diameter of the hole divided by the diameter of the hole should be greater than 1.05. No additions or changes to the "verify_data" file are needed.

The two rules are placed in the file ~/vws2/vergen2/new_hole_rules, which looks exactly like the following five lines (including the spaces between the lines).

```
The square root of the number of threads per inch
should be less than 5.0.

The countersink_diameter of the hole divided by the
diameter of the hole should be greater than 1.05.

end
```

Then the user would enter the vws2 LISP environment and make the function call:

```
(define_verifier 'hole '~/vws2/vergen2/new_hole_rules 'new_verify_hole)
```

In about five seconds the system would complete the new verification function, which it writes as shown in Figure 8. The definition is placed in the LISP environment automatically. If the user wants to save it, it may be printed to a file (most easily by a call to the LISP pretty-printer).

5. AUTOMATIC PROGRAMMING SYSTEM SOFTWARE

The software for automatically generating functions which verify features is 12 functions in the "vergen2" subdirectory of the vws2 directory. Unlike the feature verifiers themselves, which may be two or three pages long, none of the generating functions is longer than half a page. LISP appears to be particularly well suited to this type of work. In the same directory are the 9 files of rules for the nine feature types. The rules are given in [K&J2].

6. STRENGTHS AND WEAKNESSES

The automatic generation of feature verifiers has several strengths. First, it makes the feature verification system self-documenting. The rules which are implemented by the LISP code are exactly the ones given in the rules file for the feature type. These are written in plain English, readable by any user who understands the features, their parameters, and a few extra descriptive phrases.

Second, it makes the system flexible; verification rules can be changed easily. As long as the rules follow the syntax set for them, LISP code will be generated quickly, correctly, and effortlessly to implement them.

Third, it saves a lot of work for the system developers. The design protocol for the VWS2 system has changed continually for the past two years. We have found that when a feature type is redefined, the easiest part of the system to change is the feature verifiers -- because we don't have to write the LISP code. The work that went into the automatic programming system has already been paid back in the time it has saved in writing verification functions.

Fourth, it writes good LISP code. The automatic programming system works to ensure that the code it writes is efficient. Although many local variables may be given in the "features" data, only those local variables whose English equivalents appear in at least one rule are used in the verification function written by the system.

The system has at least two significant weaknesses. First, the syntax of the rules is tightly constrained. Only a few types of sentences are legitimate, and within sentences, only pre-declared words and phrases are allowed. A user unfamiliar with computer programming might be deceived by the fact that the input is English into thinking the system is more capable than it is. Also, adding words, phrases, and their LISP equivalents requires detailed knowledge of the system.

Second, the system does not make it easy to add words and phrases whose meaning requires complex calculations and the use of several variables. For example, the rule “The outline of the contour_pocket should not intersect itself.” was added to the contour_pocket verification function after it was written by the system. Adding it to the database so that it would have been handled automatically would have been more difficult.

XIV. **LIMITATIONS**

1. INTRODUCTION

Although the VWS2 verification subsystem is very extensive, it has many limitations.

2. PHYSICAL OBJECT DATA NOT CHECKED

The limitation with the greatest importance to machining is the requirement that the input data describing physical objects (workpiece, tools, fixturing, etc.) must be correct, and a great deal of data is required. It is no easier to get this data right than it is to get a design or process plan right, but there are no checks on this data. Fortunately, this data on physical objects changes slowly compared with the changes in designs and process plans. We have not made frequent changes in tooling. Sensory devices in the workstation itself would be the best method of verifying input data. The existing probing capability of the milling machine could be used to check workpieces and fixturing. A vision system which made silhouettes of tools might be used to verify tooling data. This is within the limits of the capabilities of current vision systems.

3. MILLING AREA SIMPLIFICATIONS

See Figure 3. The simplifications made in the representation of obstacles in the milling area cause some false error messages to be generated, but these are easy to deal with. A more serious problem with the milling area representation is the assumption that no part of the fixturing is above the top of the workpiece. This assumption precludes automatically checking the fixturing of a part on a pallet.

4. TOOL GEOMETRY SIMPLIFICATIONS

See Figure 5. Stopping the model at the base of the tool holder is a serious limitation. Because the model stops there, the requirement that the total depth of a cut must be less than the exposed length of the tool has been imposed on the system. This results in too many unnecessary error messages. The model should be extended to include the tool holder and (possibly) the spindle of the milling machine.

5. TWO-AND-A-HALF-DIMENSIONAL MODELLING

The entire VWS2 system is built with two-and-a-half-dimensional modelling. This is a serious limitation on the system that is not particular to the verification subsystem.

6. PRE-ENHANCEMENT DESIGN VERIFICATION NOT DONE

The type checking that takes place during design verification is performed on the design after it has been enhanced. This means that some types of design errors will not be detected until the enhancement subsystem crunches into a LISP error. It would be useful to have a second round of type checking that takes place before enhancement to catch errors of this sort.

REFERENCES

[JUN]
Jun, Jau-Shi; "The Vertical Machining Workstation Systems"; NBSIR 88-3890; National Bureau of Standards; 1988; 65 pages.

[KRA1]
Kramer, Thomas R.; "Process Plan Expression, Generation, and Enhancement for the Vertical Workstation Milling Machine in the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 87-3678; National Bureau of Standards; 1987; 56 pages.

[KRA2]
Kramer, Thomas R.; "Process Planning for a Milling Machine from a Feature-Based Design"; Proceedings of Manufacturing International Meeting; Atlanta, Georgia; April 1988; ASME; 1988; Vol. III, pp. 179 -189.

[KRA3]
Kramer, Thomas R.; "The Graphics Subsystem of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3783; National Bureau of Standards; 1988; 27 pages.

[KRA4]
Kramer, Thomas R.; "Data Handling in the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3763; National Bureau of Standards; 1988; 62 pages.

[KRA5]
Kramer, Thomas R.; "The vws_cadm User Interface in the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3738; National Bureau of Standards; 1988; 110 pages.

[K&J1]
Kramer, Thomas R.; and Jun, Jau-Shi; "Software for an Automated Machining Workstation"; Proceedings of the 1986 International Machine Tool Technical Conference; September 1986; Chicago, Illinois; National Machine Tool Builders Association; 1986; pp. 12-9 through 12-44.

[K&J2]
Kramer, Thomas R.; and Jun, Jau-Shi; "The Design Protocol, Part Design Editor, and Geometry Library of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3717; National Bureau of Standards; 1988; 101 pages.

[KR&S]
Kramer, Thomas R.; and Strayer, W. Timothy; "Error Prevention in Data Preparation for a Numerically Controlled Milling Machine"; Proceedings of 1987 ASME Annual Meeting; ASME; 1987; PED-Vol. 25; pp. 195 -213.

[KR&W]
Kramer, Thomas R.; and Weaver, Rebecca E.; "The Data Execution Module of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3704; National Bureau of Standards; 1988; 58 pages.

[LOVE]
Lovett, Denver; "The Vertical Workstation's Equipment Controllers"; NBSIR 88-3769; National Bureau of Standards; 1988; 59 pages.

[NA&J]
Nakpalohpo, Ibrahim; and Jun, Jau-Shi; "Automated Equipment Program Generator and Execution System of the AMRF Vertical Workstation"; not yet published; 1987; 17 pages.

[RUDD]
Rudder, Frederick; "Operations Manual for the Automatic Operation of the Vertical Workstation"; NBSIR 89-4031; National Bureau of Standards; 1989; 33 pages.