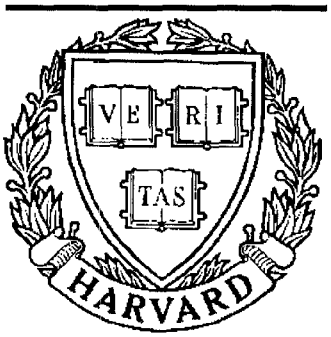
THESIS REPORT
Ph.D.

S Y S T E M S
R E S E A R C H
C E N T E R



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

An Algebraic Approach to Feature Interactions

by R.R. Karinthi
Advisor: D. Nau

Ph.D. 90-8

CS-TR-
SRC-TR-

December, 1990

An Algebraic Approach to Feature Interactions*

Raghu R. Karinthi†
Computer Science Department
University of Maryland
College Park, MD 20742

Abstract

Planning the manufacture of machined parts requires a great deal of geometric reasoning. One of the key steps in this task is the derivation of machinable features from a solid model of the part. Regardless of the approach used for obtaining the features, geometric interactions among the features can create situations where there are several possible feature representations for the same part. This presents a problem in planning the manufacture of the part, since some of these representations may be manufacturable and some may not.

This dissertation presents an automatic way of computing alternate feature interpretations using an algebra of feature interactions. The algebra is intended to enable automated process planning systems to decide whether various interpretations of a part as a collection of machinable features are feasible for manufacturing, and among the feasible ones, which is the most appropriate one for manufacturing. Alternate feature interpretations are computed by performing operations in the algebra. Furthermore, various provable properties of the feature algebra aid in resolving several of the feature interactions without even applying the operations in the algebra.

The operations in the algebra are defined on the set of all compact, regular, semi-analytic solids. A restricted subset of the algebra has been implemented in a geometric reasoning system, for use with the Protosolid solid modeler and the EFHA process planning system (which were developed earlier at the University of Maryland). We have shown through experiments that computing the operations of the feature algebra as developed in this dissertation is much more efficient than computing them by converting them to queries to a solid modeler.

This work was supported in part by an NSF Presidential Young Investigator award for Dr. Nau with matching funds from Texas Instruments and General Motors Research Laboratories, NSF Grant NSFD CDR-88003012 to the University of Maryland Systems Research Center, NSF Equipment grant CDA-8811952 and NSF grant IRI-8907890, and a Semester Research Award from the University of Maryland General Research Board.

Abstract

Title of Dissertation: An Algebraic Approach to Feature Interactions
Raghu Ram Karinthi, Doctor of Philosophy, 1990
Dissertation directed by: Dana Nau, Associate Professor, Computer Science

Planning the manufacture of machined parts requires a great deal of geometric reasoning. One of the key steps in this task is the derivation of machinable features from a solid model of the part. Regardless of the approach used for obtaining the features, geometric interactions among the features can create situations where there are several possible feature representations for the same part. This presents a problem in planning the manufacture of the part, since some of these representations may be manufacturable and some may not.

This dissertation presents an automatic way of computing alternate feature interpretations using an algebra of feature interactions. The algebra is intended to enable automated process planning systems to decide whether various interpretations of a part as a collection of machinable features are feasible for manufacturing, and among the feasible ones, which is the most appropriate one for manufacturing. Alternate feature interpretations are computed by performing operations in the algebra. Furthermore, various provable properties of the feature algebra aid in resolving several of the feature interactions without even applying the operations in the algebra.

The operations in the algebra are defined on the set of all compact, regular, semi-analytic solids. A restricted subset of the algebra has been implemented in a geometric reasoning system, for use with the Protosolid solid modeler and the EFHA process planning system (which were developed earlier at the University of Maryland). We have shown through experiments that computing the operations of the feature algebra as developed in this dissertation is much more efficient than computing them by converting them to queries to a solid modeler.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The world of physics, that is the whole of the philosophy of nature is nothing but geometry.
—Réné Descartes (1596–1650)

The ability to reason with the geometry of a solid object plays a significant role in a variety of domains. Geometric reasoning occurs in everyday events such as driving and walking, as well as in specialized areas such as various branches of engineering and medicine. However, automating this capability appears to be a difficult problem and has offered tough challenges even in the specific domains in which it has been attempted. One such domain is that of the manufacture of metal parts.

Machining a metal part consists of taking a piece of stock and performing various *processes* or *operations* on it to transform it into a desired form. Each machining process acts on a piece of stock and changes its structure in a unique way. Given a description of the object that is produced by computer aided design (CAD), one would like to derive how it can be machined. This means arriving at a sequence of processes, which when applied to the stock, would result in the final machinable part, along with other information pertaining to fixturing the workpiece, feed rates, cutting speeds, etc. Notice that such a sequence is not unique. Traditionally the designer comes up with the CAD description of the object to be machined, and the process engineer and the Numerical Control (NC) programmer produce the precise process plan.

The fields of computer aided design and computer aided manufacturing (CAM) have made considerable progress in the last 15 years; the former concentrating on solid modelers, part specification protocols and drafting aids, and the latter concentrating on the development of generative and variant process planning systems.

Many of the problems faced by modern industry are related to a lack of coordination between design and manufacturing. Typical problems include inconsistencies among process plans for similar designs, large discrepancies from optimal shop utilization, poor product quality, and non-competitive costs. One of the ways to remedy these problems is to take the manufacturing issues such as the availability of tools, jigs and fixtures into account during the design process rather than afterwards. This integrated approach to design and manufacturing is known as *concurrent engineering*. This requires that CAM, modules such as process planning systems, interact

closely with CAD systems. It will be necessary for process planning systems [Cha90] to reason about geometric relationships among the various parts of an object while the design is underway. But the achievement of such interaction presents several unresolved problems. One of the primary problems is that generative process planning systems consider a machinable part to be a collection of machinable features, and it is necessary to obtain descriptions of such features from the CAD representation. Several approaches have been proposed for doing this, as discussed below.

1. *Automatic feature extraction* focuses on extracting manufacturing features from existing CAD databases such as IGES files, BReps, etc. Several researchers have addressed this problem using syntactic pattern recognition, rule-based systems, graph-based approaches or combinations of the above. Prominent among these are the ones by Henderson [Hen84], Kyprianou [Kyp80], De Floriani [DF89], Kumar[Kum88], Joshi[Jos87] and Srinivasan [SL87].

   Some of the more significant problems with feature extraction are as follows:

   (a) Certain attributes of a machinable part cannot be made without reference to a particular feature (for example, the surface finish, corner radius, and machining tolerances of a pocket). When an object is designed without making reference to these features explicitly, it is unclear how to associate the machining specifications with the proper features.

   (b) It is difficult to extract a feature which intersects or otherwise interacts with other features, without disturbing those other features. For example, in Henderson's feature extraction system [Hen84], once a feature volume has been recognized it is subtracted from the overall cavity volume—making it impossible to obtain multiple feature interpretations for the same cavity volume.

2. In *design-by-features*, the user builds a solid model of an object by specifying directly various form features which translate directly into the relevant manufacturing features. Systems for this purpose have been built for designing injection-molded parts [VDZS85], aluminum castings, [LDS86], and machined parts [KJ87, Ide87, BH87, CT87, SR88].

   In the case of machined parts, one problem with design by features is that it requires a significant change in the way a feature is designed. Traditionally, a designer designs a part for functionality, and a process engineer determines what the manufacturable features are. However, design by features places the designer under the constraints of not merely having to design for functionality, but simultaneously specifying all of the manufacturable features as part of the geometry—a task which the designer is not normally qualified to do. Another problem—that of alternate feature interpretations—is described in Section 1.1.

3. *Human-supervised feature extraction* overcomes one of the problems of design by features, by allowing the designer to design the part in whatever way is most convenient, and then requiring the process engineer to identify the machinable features of the part. A system for this purpose was built at General Motors Research Laboratories, and another is being built at the National Institute of Standards and Technology [BR87] using Unicad/Romulus [ROM85].

   Human-supervised feature extraction provides a way for a qualified manufacturing engineer to identify the machinable features–but it still does not handle the problem of alternate feature interpretations. Sometimes a machinable part can be specified as a set of machinable features in more than one way—and unless one is very careful to identify all such possible interpretations, this can lead to significant problems in process planning.

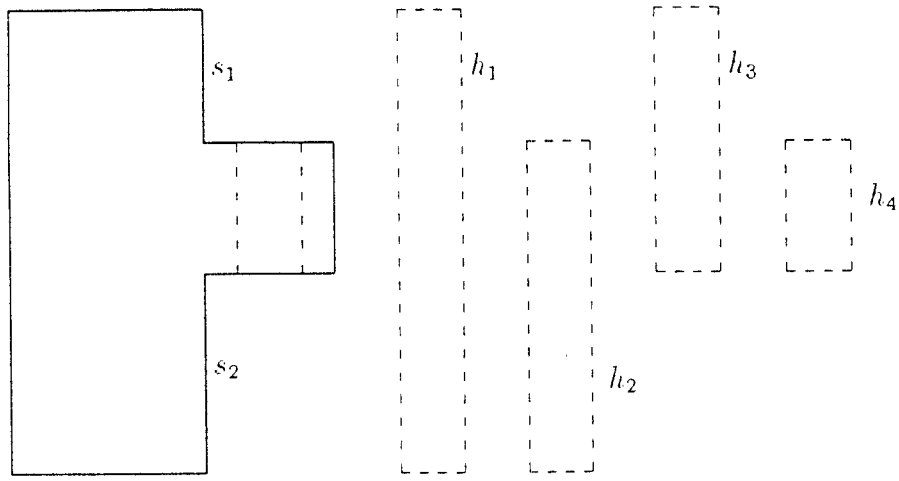


Figure 1.1: A part with two slots and a hole.

1.1 Geometric Interactions Among Features

Consideration of the above approaches makes it evident that regardless of the approach used for designing a part, a major problem to be solved is handling feature interactions. Geometric interactions among the features can create situations where there are several possible feature representations for the same part. To produce a good process plan—or, in some cases, even to produce a process plan at all—it may be necessary to use a different interpretation of the part than the one obtained from the CAD model. The problem is how to find these alternate interpretations. The importance of handling feature interactions has been stressed in the recent reports such as [SRSM89, Rog89]. Chapter 2 discusses related work addressing feature interactions at a greater detail.

Let us consider a few examples to show the role of geometric feature interactions.

1. Consider the part shown in Figure 1.1. In this example, the part has been described as the part resulting from subtracting a hole h_1, a slot s_1 and a slot s_2, in that order out of a rectangular stock. But because of the interaction of h_1 with s_1 and s_2, we get $h_2 = h_1 -^* s_1$, $h_3 = h_1 -^* s_2$, and $h_4 = h_2 -^* s_2 = h_3 -^* s_1$. The final set of features used for machining would be s_1, s_2 and one of the holes h_1, h_2, h_3 and h_4. Which hole to use depends on issues such as cost criteria (whether it is cheaper to make a deep hole or a small hole), feasibility (availability of proper tools to make a deep hole), fixturing criteria (whether it is possible to fixture the part to prevent excessive vibration while making h_4 after s_1 and s_2 have been made), and machinability criteria (whether vibration during the machining of h_4 will allow acceptable machining tolerances to be achieved, or whether h_1 can be machined with an acceptable straightness tolerance). Thus, one can see that there can be several possible interpretations such as $\{h_1, s_1, s_2\}$, $\{h_2, s_1, s_2\}$, $\{h_3, s_1, s_2\}$ or $\{h_4, s_1, s_2\}$ for the same part, and having only one of them can be a serious limitation in process planning.

2. As an example, consider the object shown in Figure 1.2. A detail of a portion of this object is shown in Figure 1.3. Due to geometric interactions among the features h_1, h_2 s_1, s_2, s_3, and s_4, there are several interpretations as to their identities. For example, as shown in Figure 1.4, there are eight different interpretations of h_1. Each interpretation corresponds to a different order in which h_1, s_1, s_2, and s_3 are made. For example, in interpretation (a), h_1 is made

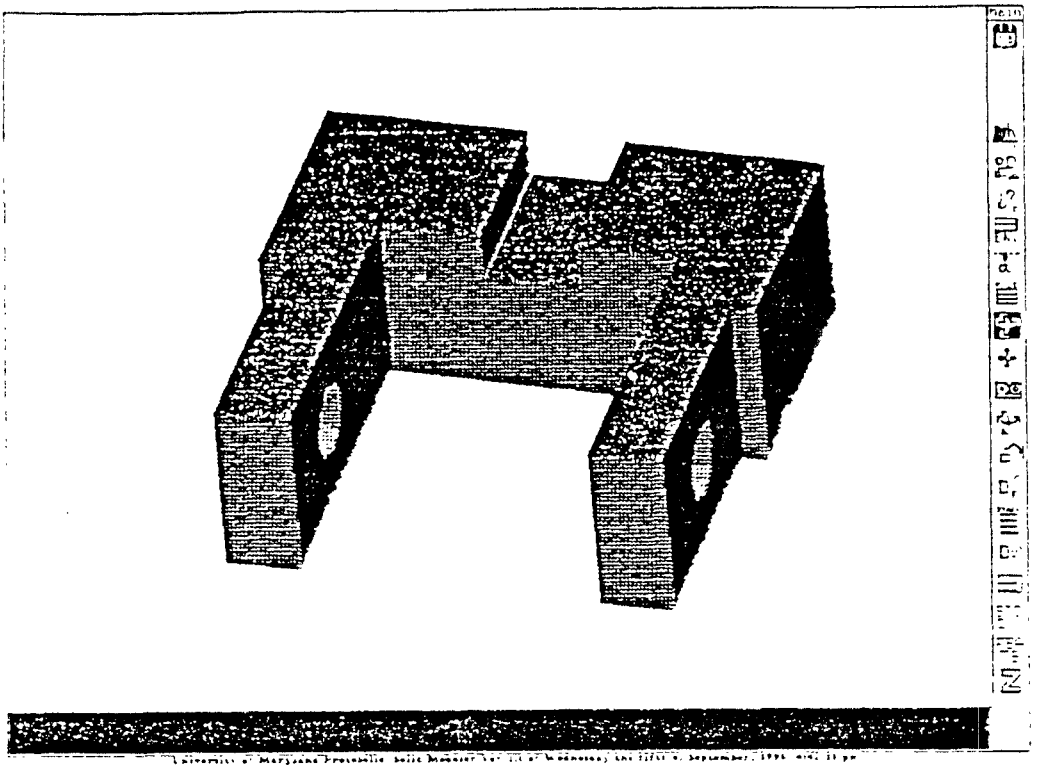


Figure 1.2: A bracket for use with a bearing. This image was produced using the feature algebra interface and the Protosolid solid modeler.

after s_1, s_2, and s_3, but in interpretation (b), it is made before s_1.

Depending on the widths W_1, W_2, and W_3 and the concentricity tolerance C, one or another of the interpretations may be best one in terms of manufacturability. For example, if C, W_1, W_2, and W_3 are all small, then interpretation (h) is best. But if C, W_1, W_2, and W_3 are all large, then interpretation (a) is best. For other values of C, W_1, W_2, and W_3, several of the other interpretations may be best.

3. Consider the part described in Figure 1.5. The part has two features: an angle a_1 and a hole h_1 as described by the designer. However, one cannot drill on a slanted surface, due to the drill bit slippage problem. Several things can be done to prevent the problem. If the slant surface is also machined, one can drill the hole before milling the slant surface. However, this would mean drilling a deeper hole h_2. The other approach is to make the slant surface first and then machine the hole h_1. Thus, we have two feature interpretations $\{h_1, a_1\}$ and $\{h_2, a_1\}$. If there is a tight parallelism tolerance between the axis of the hole and the bottom surface of the part, one has to choose the interpretation $\{h_2, a_1\}$. However, if the parallelism constraint is not strict and if the angle θ is small one chooses the interpretation $\{h_1, a_1\}$. Thus, given either feature interpretation of the part, one needs the capability to generate the other.

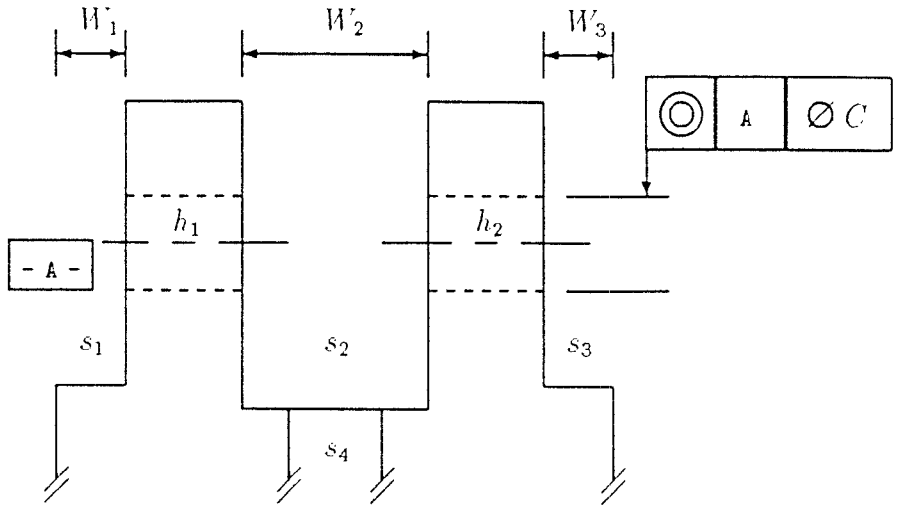


Figure 1.3: A portion of the bracket shown in Figure 1.2. W_1, W_2, and W_3 are the widths of s_1, s_2, and s_3, respectively, and C is the concentricity tolerance for the holes h_1 and h_2.

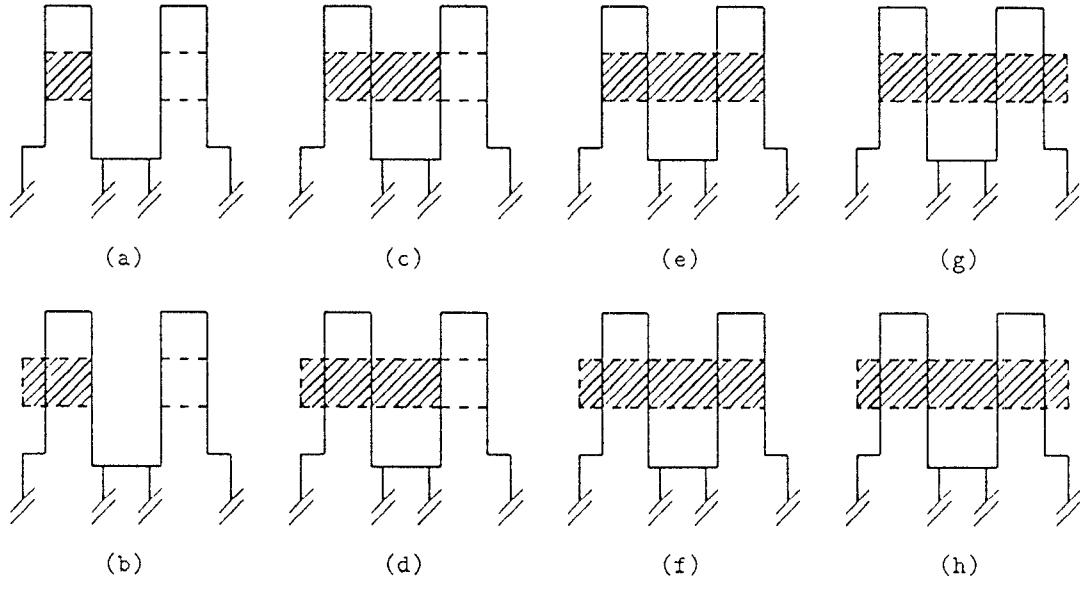


Figure 1.4: Different interpretations of the hole h_1.

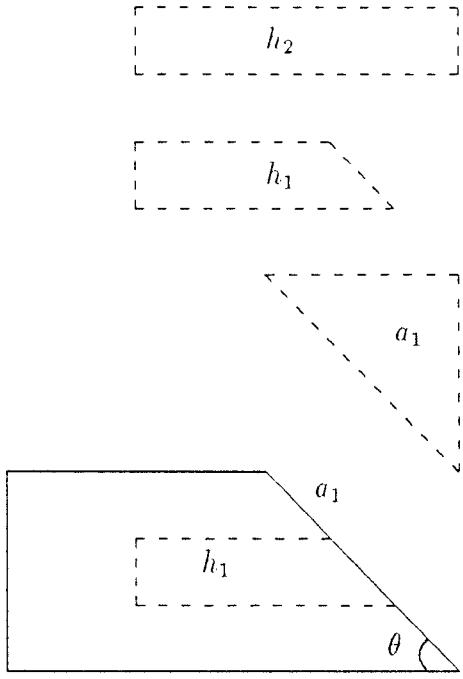


Figure 1.5: A part with an angle and a hole.

1.2 Motivation

Given the importance of feature interactions, a major goal was to develop an approach towards handling them. The following issues were considered in developing an approach:

1. The approach should be powerful enough to handle a wide variety of features and geometric interactions among features.

2. The approach should be expressible clearly and precisely. Developing a mathematical approach is one way to achieve this.

3. The approach should be simple and elegant; it should capture the interactions in a unified way rather than as a number of special cases.

4. The approach should be implementable efficiently using existing hardware and existing software tools such as programming languages, and solid modelers.

1.3 Overview of the Algebraic Approach

This thesis discusses an approach based on an algebra of features for handling feature interactions. The feature algebra is an algebraic structure which consists of a domain of subtractive features and a collection of operations that are defined on pairs of features. In addition, certain algebraic properties of the operations in the feature algebra have been proved. By performing operations in the algebra one can find alternative feature interpretations of a part. We have developed an algorithm that generates all the feature interpretations of a part, given one feature interpretation. This algorithm uses the operations and the algebraic properties.

Using this framework, a subset of this feature algebra dealing with rectangular solids, cylinders and countersinks has been implemented. The feature algebra serves as the basis of a geometric reasoning system for use in communicating between the Protosolid solid modeler [Van89b] and the EFHA process planning system [Tho89]. Protosolid is a boundary representation based solid modeler developed at the University of Maryland, College Park. EFHA (Environment For Hierarchical Abstraction) is a process planning system also developed at the University of Maryland, College Park. Both Protosolid, EFHA and the feature algebra run on a Texas Instruments Explorer II[1] and are written in Common Lisp [Ste84], except for the routines for interactive and graphical display which use object-oriented programming techniques such as flavors, the Explorer window system, and other features specific to the Explorer series of machines.

The brief workings of the feature algebra as implemented on the Explorer are as follows:

The designer describes the part by starting with a piece of stock and then successively subtracting features out of the stock. After all the features have been specified, the designer invokes the algorithm which generates alternative feature interpretations of the part. The algorithm also produces the input suitable to do process planning and communicates them to EFHA process planning system. EFHA comes up with the least cost process plan for a feature (if it exists). Thus, the final output of the system is all feature interpretations of a part and optionally, the process plans for all the features in all the feature interpretations.

1.4 Main Contributions of the Thesis

The contributions of this thesis are discussed in detail in Section 8.1. In this section, we summarize the main contributions of this thesis. In this thesis, an algebra of features has been developed for producing multiple feature interpretations of a machinable part. The representation of features includes not only the shape of the features, but also information about their boundary useful for process planning. Operations on features and algebraic properties of the operations have been developed. A subset of the algebra has been implemented and has been shown to be efficient for producing multiple feature interpretations of a part. The sub-algebra that has been implemented has been integrated with a solid modeler and a process planner.

1.5 Organization of the Thesis

This thesis is organized as follows: Chapter 2 describes some of related research in the area of feature extraction and in handling feature interactions. Chapter 3 describes the algebra of feature interactions that has been developed in this thesis. In Chapter 4, we describe an algebra for a restricted class of features that has been implemented. Chapter 5 describes an interface for feature based design that that serves as a front-end to the restricted feature algebra. Chapter 6 describes some of the details of the feature algebra implementation. In Chapter 7, we illustrate the workings of the feature algebra through some examples. Chapter 8 summarizes the algebraic approach proposed in this thesis for handling feature interactions. Some ideas for further research along the lines of the proposed approach are also suggested.

[1]The keywords Explorer and Explorer II used in this thesis are trademarks of Texas Instruments Incorporated.

Chapter 2

Background

When a figure or "positive space" (e.g., a human form, or a letter or a still life) is drawn inside a frame, an unavoidable consequence is that its complementary shape — also called the "ground", or "background", or "negative space" — has also been drawn. In most drawings, however, this figure-ground relationship plays little role. The artist is much less interested in the ground than in the figure. But sometimes, an artist will take interest in the ground as well.

—Douglas R. Hofstadter, Gödel, Escher, Bach: an Eternal Golden Braid

Beginning with the use of numerically controlled machines over three decades ago, there has been considerable interest in automating the task of planning for machinable parts. One of the key steps in this process is the integration of the representation schemas suitable for computer aided design and computer aided manufacturing. As mentioned in Chapter 1, there are three popular approaches towards the integration of CAD and CAM. In this chapter, we will examine these approaches at a greater detail.

2.1 Automatic Feature Extraction

The essential task in automatic feature extraction is to derive machinable features from an existing CAD description of the part such as the Constructive Solid Geometry (CSG) tree, boundary representation (BRep) and Initial Graphics Exchange Specification [SW86] (IGES) files. However, some researchers have attempted to directly generate the numeric control (NC) tool paths without actually generating the features. Prominent among these is the work of Grayer[Gra76] and Armstrong etal.[AGP84].

Most researchers have focused on extracting features from the BRep of a part. Notable exceptions are the work by Woo [Woo82] which refers to features as volume elements (without referring to a specific representation scheme) and that of Kumar [Kum88] that extracts machinable features from an IGES representation of the part. Although extracting features from wire frame representations and engineering drawings [HQ82] is interesting because of its immediate applicability to industry, most of the researchers have preferred to work with the boundary representation, since it offers an unambiguous and complete representation of the part geometry.

Before discussing the work related to automatic generation of machinable features we will describe the work that directly generates the NC tool paths from the CAD description of a part.

2.1.1 Direct NC Code Generation

Grayer[Gra76] used the BRep of a part obtained from the BUILD[Bra79, Str80] solid modeler. The program he has developed compares the final representation of the part with the initial stock and divides the material to be removed into a set of horizontal laminae of constant cross section. The NC tool paths are generated using a postprocessor. A limitation of Grayer's work is that it can handle only pockets and holes with vertical walls.

Armstrong et al.[AGP84] have developed a way of directly generating the NC code from the BRep of the final part obtained from the PADL-1[HM85] solid modeler. The idea is to decompose the part into 3D volume elements which are used to guide the NC code generation. The program can generate the NC code for the roughing cut and the finishing cut in a vertical milling machine. This program does not generate machining features explicitly.

Machining features serve as an abstract level between the CAD description of the part and the NC code so that a shape element could be machined by one or more of a variety of processes such as end milling, center drilling or twist drilling. An attempt to directly generate the NC code for a specific machine loses this flexibility. However, this approach will be useful in cases where there is only one kind of machine available.

2.1.2 Methodologies in Feature Extraction

There appear to be three dominant approaches in extracting machinable features from CAD models:

1. Algorithmic approaches such as syntactic pattern recognition and graph based approaches;
2. Rule-based systems;
3. Combination of the above.

Syntactic Pattern Recognition

Kyprianou [Kyp80] used syntactic pattern recognition techniques to extract features from the boundary representation of a part. The goal of his research was to generate Group Technology (GT) codes for parts, based on their features. His program has the following phases:

1. marking the edges, vertices and faces of the part based on convexity / concavity information;
2. partitioning the faces into groups known as *facesets*;
3. recognition of features in a faceset;
4. recognition of the overall shape of the part;
5. GT code generation from the overall shape.

Kyprianou's program has the capability to identify depressions, protrusions and bridges. In his scheme, a depression is a connected set of faces such that each face is adjacent to at least one other face via a concave edge and all the edges bounding the depression are convex. Syntactic pattern recognition techniques are used to identify the features. Figure 2.1 below illustrates the grammar for identifying depressions. The depression grammar is of the form $G =< N, T, P, S >$, where N

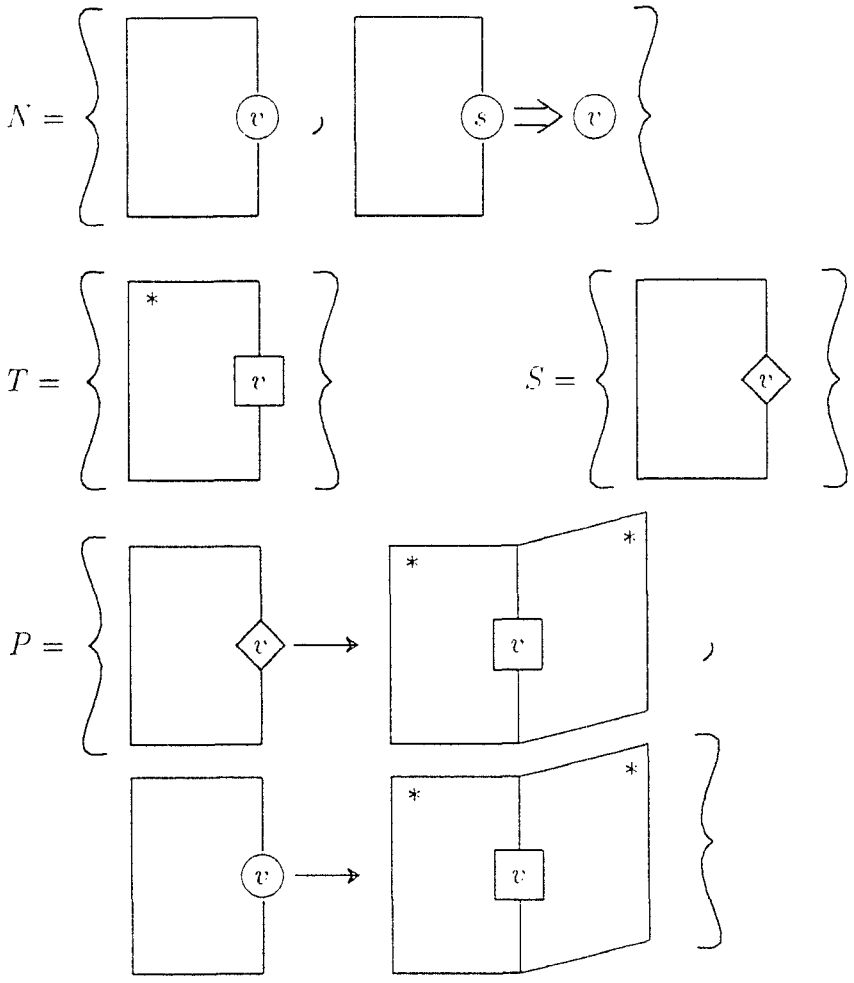


Figure 2.1: The grammar for recognizing depressions.

is the set of non-terminal primitives, T is the set of terminal primitives, P is the set of production rules and S is the starting primitive. The $*$ indicator in the depression grammar designates that a face has been included in the set of faces forming a depression and the symbol v denotes a concave edge. Figure 2.2 shows a part with a simple slot and how it is parsed by the depression grammar.

The grammar developed by Kyprianou does not have the capability to address feature interactions. Furthermore, Kyprianou's features are not volumetric. Therefore, from a process planning perspective, the volume to be machined is not defined.

Dong and Wozny [DW88] and Falcidieno and Giannini [FG89] have extended the work of Kyprianou resulting in algorithms that can recognize a wider class of features.

Jakubowski [Jak82] has developed a grammar for recognizing the external contour of a machined part. Jakubowski uses a extended context free grammar, which is similar to a context free grammar except that the right hand sides of the production rules can be regular expressions over the terminal and non-terminal symbols.

The following simplified example illustrates the kind of grammars used by Jakubowski [Jak82] and Staley et al. [SHA83] for recognizing the external contour of a part. The exact grammar would be too complicated for the purpose of illustration. Let us define four pattern primitives which are positive and negative unit vectors along the co-ordinate axes in two dimensions as shown in

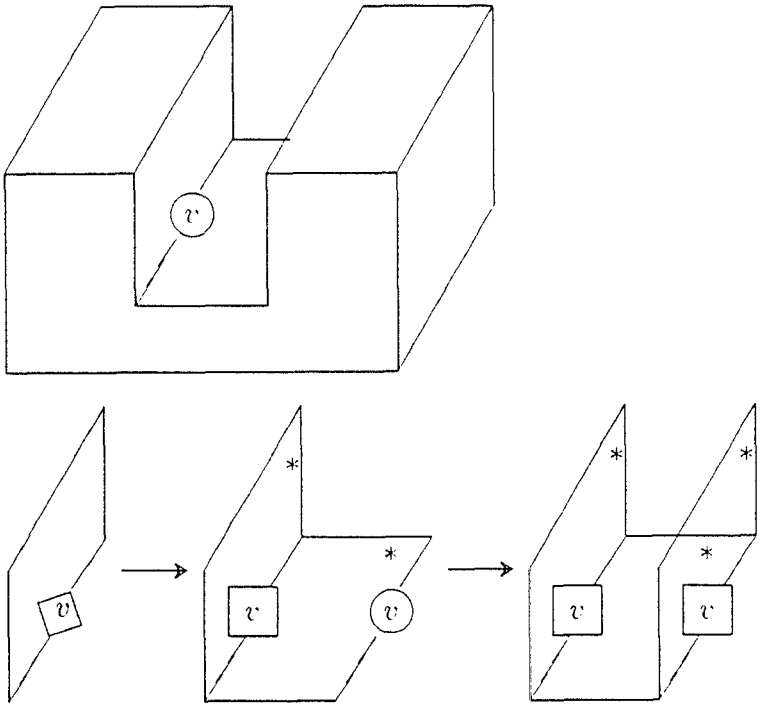


Figure 2.2: A part with a slot and a parse of the slot faces.

Figure 2.3. Let us define the following grammar using the pattern primitives. $G =< N, T. P, S >$. where S is the start symbol. T is a set of terminal symbols. N is a set of non-terminal symbols and P is the set of production rules.

$$\begin{aligned} T &= \{a, b, c\} \\ N &= \{S, A, B\} \\ P &= \{S - cAa \\ &\quad A - B|cAa \\ &\quad B - b|bB\} \end{aligned}$$

The language generated by the grammar is $L = \{c^n b^m a^n m, n \geq 1\}$. The contour of the cross section of a flat-bottomed hole, as traversed from left to right can be parsed by the grammar(see Figure 2.4).

A limitation of the grammars by Jakubowski and Staley is that they are two-dimensional in nature.

Graph-Based Approaches

Given the boundary representation of a part, a variety of graph structures can be constructed based on the adjacency relationships among the vertices, edges and faces. This approach to feature extraction looks for patterns in such graphs that correspond to features.

Leila De Floriani [DF89] has developed a technique for the extraction of protrusions, depressions, through holes and handles from the boundary representation of a solid. This technique is based on

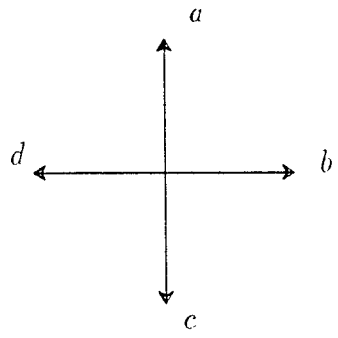


Figure 2.3: Pattern primitives for the grammar G.

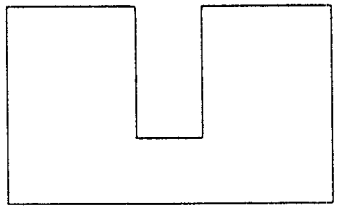


Figure 2.4: Cross section of a flat-bottomed hole.

recognizing the features from the edge-face graph of the solid. The edge-face graph is partitioned into its bi-connected and tri-connected components [Har69]. At each step, subgraphs of the edge-face graph G, describing the features are detected by considering mutual relations among the bi-connected and tri-connected components of G.

We will illustrate De Floriani's approach through an example. Figure 2.5 shows a part with a protrusion. The face F of the part contains two edge loops, an outer loop L_1 and an inner loop L_2. The edge-face graph of the part is shown in Figure 2.6. In this graph, loops are denoted by square-shaped nodes and faces by circular nodes. Every loop has an arc connecting it to the face containing the loop. Two loops have an arc between them if they share an edge. Without getting into the complexities of Floriani's algorithm, one can see that identifying the two bi-connected components of the graph break the part into a main block and a protrusion.

One of the limitations of this approach is that it considers only topological information. Topological information alone is not sufficient in extracting machinable features in certain cases. A combination of both geometry and topology is required.

Pinilla etal. [PSP89] have addressed this issue, and the graph-grammar they are developing has both geometric and topological information. Also, their scheme can model feature interactions. However, it appears to us that the number of rewrite rules required to model the diversity of interactions that occur in the machining domain will be very large.

Rule-Based Approaches

Rule-based approaches to feature recognition use IF-THEN rules to detect features of interest. Prominent among these is the work of Henderson[Hen84] and that of Kung[Kun84]. One of the drawbacks of rule-based systems, as opposed to more formal approaches such as grammars and

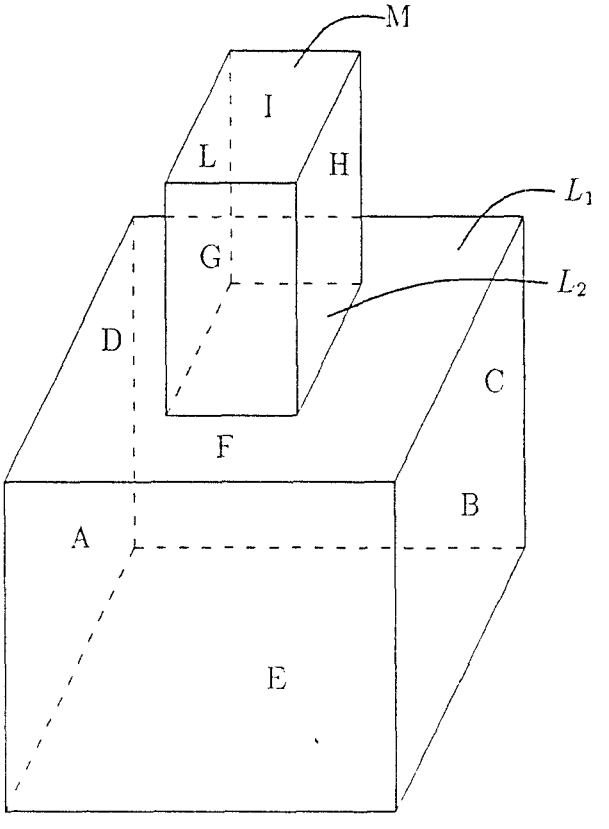


Figure 2.5: A part with a protrusion.

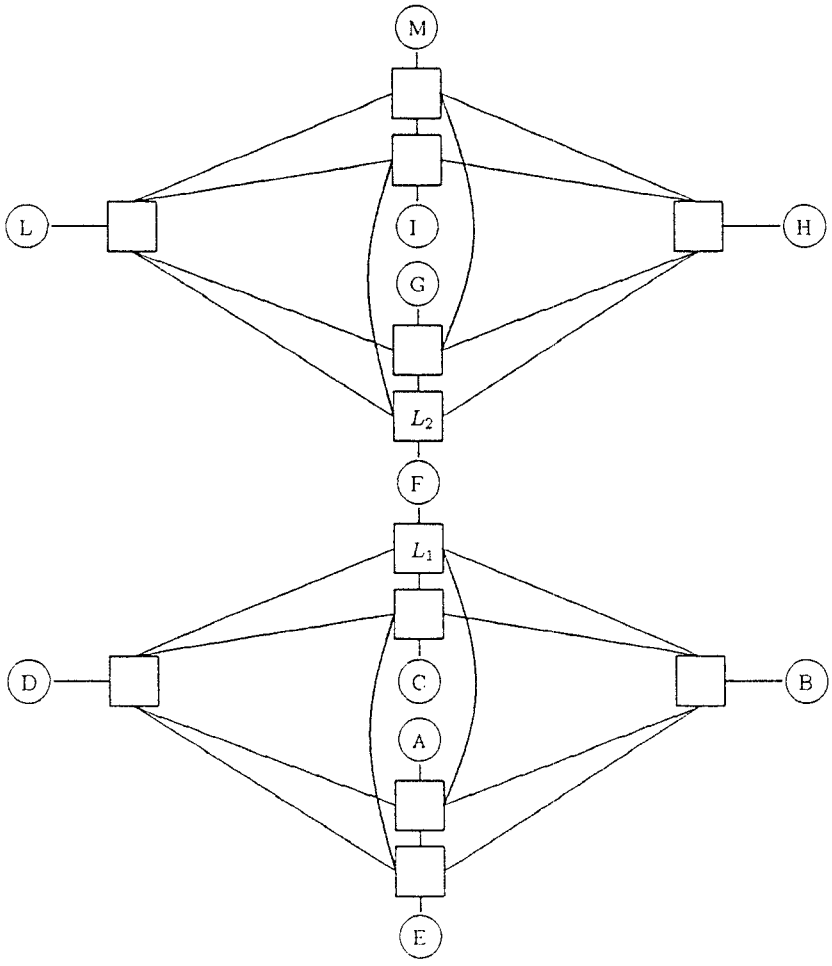


Figure 2.6: The edge-face graph of the part

graph-based algorithms, is that it is hard to characterize the domain of the features and interactions recognized by rule-based systems.

Henderson's [Hen84] system extracts features from the boundary representation of a solid obtained from the Romulus[ROM85] solid modeler. Henderson's rules for feature extraction are written in Prolog. Briefly, Henderson's feature extraction algorithm works as follows:

In the first step, his program identifies the cavity volumes (a cavity volume is a maximally connected volume that needs to be machined) in the part. The feature extraction procedure is carried out independently for each cavity volume. There could be several features in a cavity volume. Features are recognized using IF-THEN rules such as the one shown below:

> IF a hole entrance exists, and
> a cylindrical face is adjacent to the entrance and,
> there is a bottom face to the cylinder,
> THEN the entrance face, the cylindrical face and the bottom face comprise a cylindrical hole.

Once a feature has been recognized, the feature volume is subtracted from the cavity volume and features are extracted from the remaining cavity volume. This is always possible in Henderson's system because the features in his system are well-defined volume elements. This process is repeated until the cavity volume becomes empty.

Henderson's system can handle only $2\frac{1}{2}$D features. In Henderson's system, once a feature volume has been recognized it is subtracted from the overall cavity volume. Therefore, it is not possible to obtain multiple feature interpretations for the same cavity volume. The need for doing this has been pointed out by Henderson in the *Future Work* section of his thesis:

> Figure 46b [Figure 2.7 in this chapter] contains either a slot between two separate holes or a long hole intersecting a slot. In both of these cases, conflict resolution is necessary together with optimization of features based on techniques available.

Combinational Approaches

Prominent among the systems that combine algorithmic techniques and rule-based systems are those of Srinivasan and Liu [SL87] and Joshi and Chang[JC88].

Srinivasan and Liu [SL87] have developed a tree grammar (very similar to an AND/OR graph), for representing the knowledge of process capabilities. The grammar captures the shapes that can be machined using a particular process such as milling. There are two steps in his algorithm. The first step, feature recognition, involves parsing the boundary representation of a part using the grammar for a specific process. The second step, known as feature reconstruction, considers certain kinds of feature interactions (within a process model). A rule based approach is used in the feature reconstruction step. This method is more useful for capturing the geometric meaning of process capabilities rather than as a general feature extraction paradigm.

Joshi and Chang [JC88] have developed a system in which the patterns in the graph structure of a part are used to identify the features. Using the boundary representation of a part, an undirected graph, known as the attributed adjacency graph, is constructed. The vertices in this graph are the faces; and if any two faces share an edge, then there is an arc between the corresponding vertices. In addition, the arcs are marked as 0 (concave) or 1 (convex) based on whether the corresponding edge is concave or convex. Using this scheme, one can identify the patterns that characterize some common features such as the step, the slot, the corner step, the notch and the pocket (see Figure 2.8).

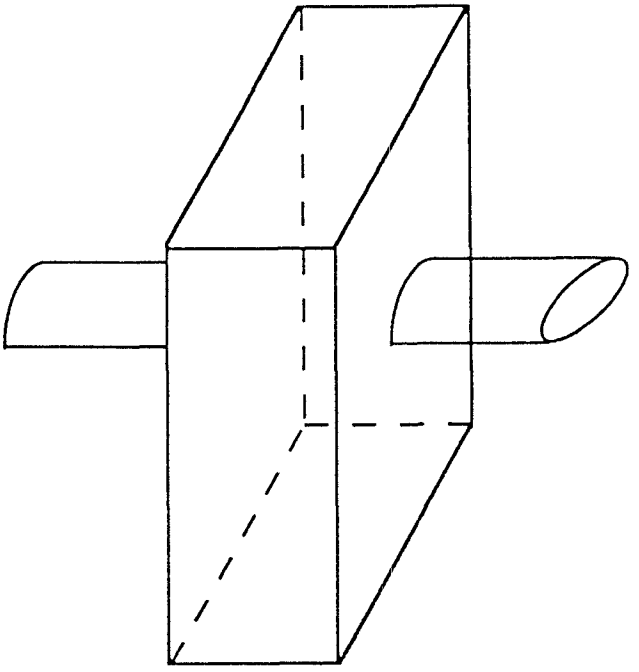


Figure 2.7: An example of a hole intersecting a slot.

Comparing this scheme to that of De Floriani the following observations may be made:

1. De Floriani's scheme relies purely on topological information. Therefore, in order to recognize depressions or protrusions on three or more faces, the boundary description of each feature must be suitably completed with dummy faces, edges and vertices to form a feature volume. The technique by Joshi and Chang is able to recognize such features by using a little geometric information, viz., convexity and concavity.

2. The geometric information captured by Joshi's attributed adjacency graph is not adequate to uniquely characterize the machining features. For example, the attributed adjacency graph does not have the information required to distinguish a dovetail slot from a right-angled slot. It also does not provide the information required to handle feature interactions. Joshi and Chang developed IF-THEN rules to handle such situations. Figure 2.9 shows an example where a step has been split into two by a slot. Here, rules are used to recognize that the corresponding faces of the two steps are coplanar and hence can be combined into a single step.

2.1.3 The Convex-Difference Method

Woo[Woo82] proposed an algorithm that uses the convex hull of a part in feature extraction. The convex hull of a solid is the minimal convex set enclosing the solid. To begin with, the convex hull of a part is computed, and this is assumed to be the raw stock. The idea is to rewrite a given solid as a finite alternating (regularized union and difference) series of convex solids. Denoting the convex hull operator by C, the procedure is illustrated in Figure 2.10. The main drawback with this approach however, is that the shapes produced for features do not necessarily correspond

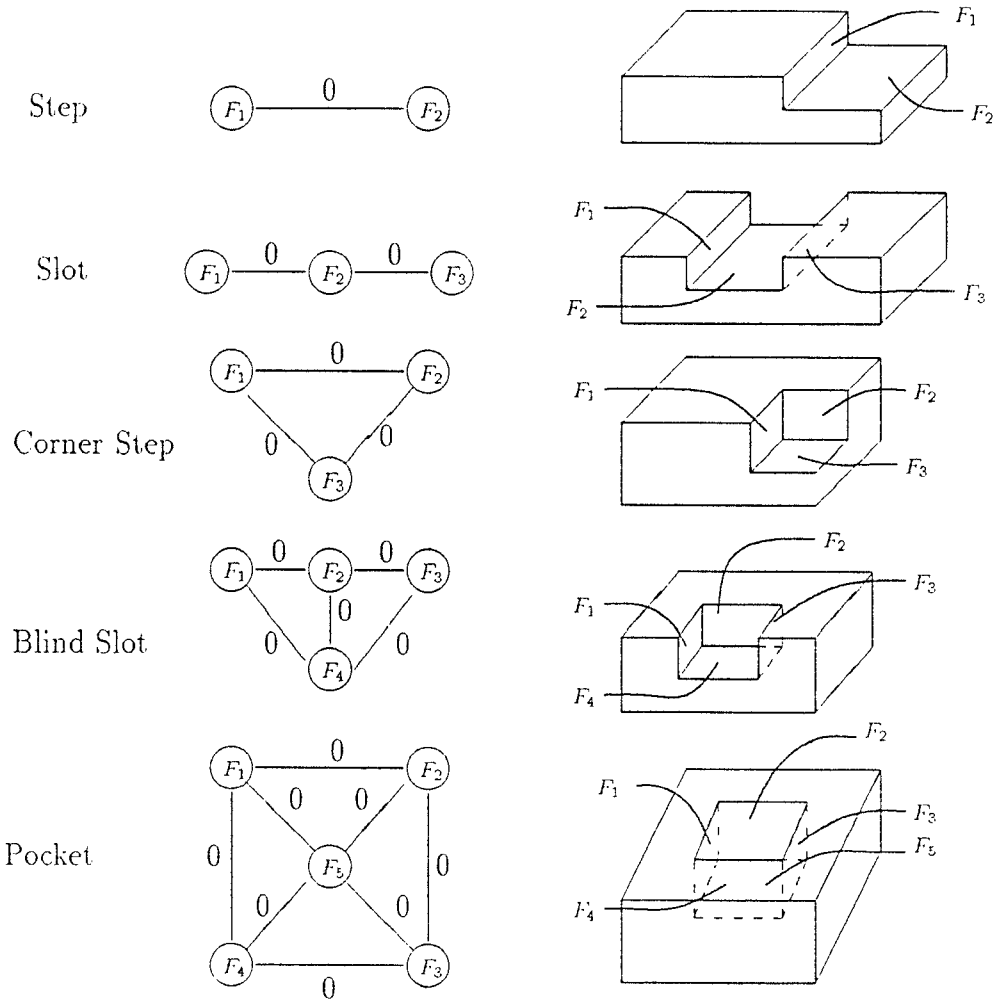


Figure 2.8: Features represented by the AAG.

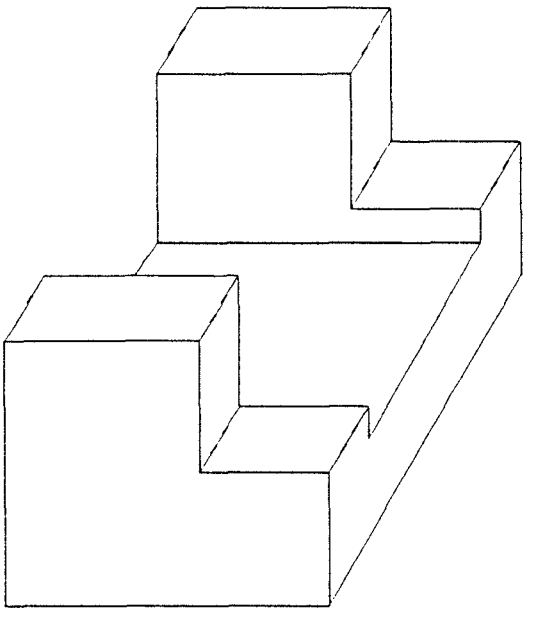


Figure 2.9: A step split into two by a slot.

to manufacturing features. A second drawback is that there can be pathological cases where the algorithm does not terminate.

In Figure 2.10, we can write the final part D_0 as

$$D_0 = H_0 - H_1 + H_2 - H_3$$

where $-$ denotes regularized subtraction and $+$ denotes regularized union. If the only features we have are depressions, then we can rewrite the above equation as a sequence of solids subtracted from the convex hull of the part (H_0), and each of these solids corresponds to a feature in the part. Therefore,

$$D_0 = H_0 - (H_1 - H_2) - H_3$$

and $H_1 - H_2$ and H_3 are the two features in the example part.

2.2 Design-by-Features

In the recent years, design-by-features systems have been developed for a variety of domains. This section discusses design-by-features systems pertaining to the domain of machinable parts only.

Kramer and Jun [KJ87] at the National Bureau of Standards (NBS, now known as the National Institute for Standards and Technology, NIST), Automated Manufacturing Research Facility (AMRF) have developed a design specification protocol called VWS2 to specify both geometric and non-geometric information of machine parts. VWS2 can handle parts which are one-sided, two and a half dimensional and can be made by subtracting features from a rectangular block of stock. The part editor provides a convenient way of specifying feature parameters. The VWS2 displays the part design on the screen. It can be used to generate process plans and to produce NC code which can be downloaded onto a machine tool to produce a part.

XCUT is an expert system developed at Bendix Kansas City Division for producing automated process plans. Hummel and Brooks [HB86] discuss the symbolic representation of manufacturing

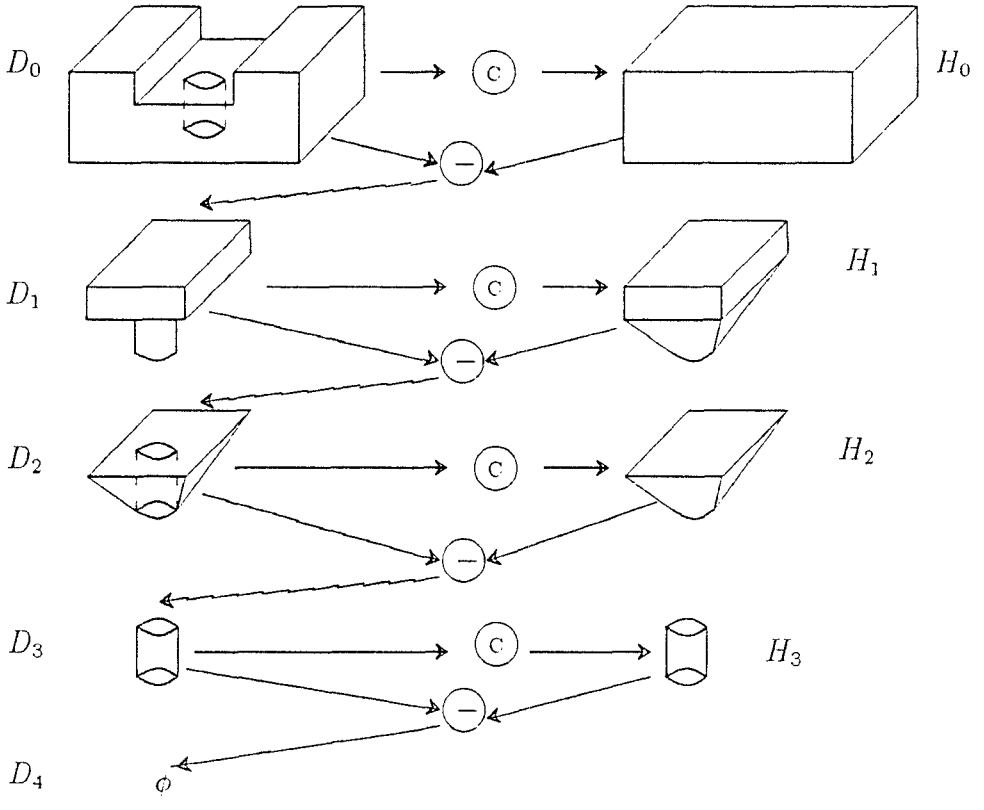


Figure 2.10: The convex hull approach due to Woo.

features in XCUT, which is organized into a taxonomic hierarchy. Inheritance of properties through the hierarchy enables elegant specification of the features. The features thus specified are used for automated process plan generation. All the geometric and topological information for a feature is extracted from the solid model of the part using the Romulus solid modeling system [ROM85] and an interactive interface. This interface allows the user to manually identify the features using a graphical pointing device. Tolerances of size and location are entered manually once the geometry and topology of the feature have been defined.

The two systems discussed above are fairly complete in the sense that they translate from part specifications to low-level process plans. But they require the designer to specify the precise manufacturing features to be planned for. The design by features approach is also used by an integrated process planning system called FIRSTCUT [TC89] being developed at the Stanford university.

2.3 Human-Supervised Feature Extraction

In trying to develop an integrated approach to design and process planning, researchers have realized that a completely automated system may not be feasible in the near future, and may not even be desirable since formal description languages for various phases of automated manufacturing have not yet been defined. Therefore, there is some interest in developing tools that automate only some of the steps in design and manufacture, and require human intervention for the remaining steps.

The work in the Automated Manufacturing Research Facility (AMRF) at the National Institute for Standards Technology [BR87, BR88] is an example of such research. Given a description of the part, a process engineer identifies the features using a computer interface. In addition, the process engineer can also specify the precedence relations among the features in the form of a graph. This information is used later by the modules for process selection, fixture planning, etc.

2.4 Work Addressing Feature Interactions

This section summarizes some of the work directed at handling interactions among the features.

Hayes [Hay87] has built a program called *machinist*, which captures the knowledge a machinist uses in process planning. This program has the capability to identify feature interactions. The knowledge for detecting the interactions is built into rules written in OPS5. Using its rules the machinist program derives precedence relations among the features.

For example, consider the part shown in Figure 2.11. It has five features viz., a shoulder, an angle and three holes named hole1, hole2 and hole3. The program identifies the following interactions:

1. The first interaction is between the angle and hole3. If the angle is made first, it will interact with the hole, by causing the drill bit to slip on the slanted surface. This will make the hole placement inaccurate. The restriction that this interaction imposes is that hole3 must be made before the angle.

2. The second interaction is between hole3 and the shoulder. If the shoulder is made first, the part will be too thin and floppy when it is clamped to cut the hole. Therefore, the hole3 must be made before the shoulder.

3. For similar reasons, the angle must be made before the shoulder.

From the above precedence relations a partial order is established among the features. This is subsequently used in process planning.

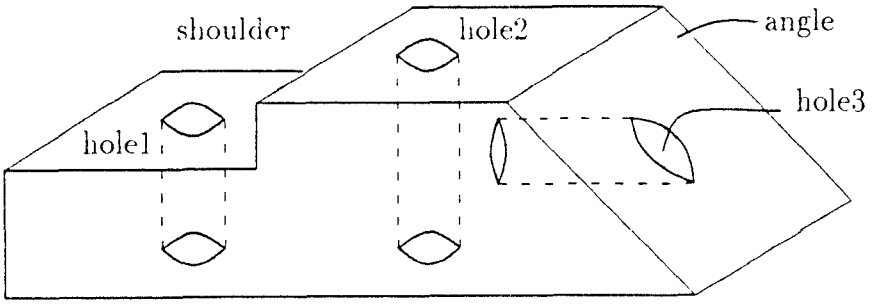


Figure 2.11: A part with three holes, a shoulder and an angle.

The machining expertise captured by the *Machinist* system is very useful. However, its representation of a solid is not adequate for process planning purposes. For example, in the *Machinist* representation, hole3 (Figure 2.11) and hole3 extended into the angle are considered to be the same hole. Thus, the depth of a hole is not considered by this system, and yet such information will be required during process planning. The machinist system will be of greater utility when it is integrated with a solid modeler.

Micahel Pratt [Pra87] has developed the notions of *canonical feature volume*, *attached feature volume*, *effective volume of intersection* and *actual volume of intersection* to capture the interactions among features. Pratt's work addresses the interactions among positive and negative features. The canonical feature volume (CFV) is the region of space occupied by a volumetric feature as evaluated from a generic description in terms of specified dimensional parameters, as correctly positioned and oriented with respect to the part model. The attached feature volume (AFV) is the region of space common to a positioned and oriented canonical feature volume and the body of the part to which the feature is attached. The effective volume of intersection (EVI) of two interacting features is the boolean intersection of the AFV of the first feature defined with the CFV of the intersecting feature. Pratt has developed a graph structure where the relationships among the features are shown. However, it is not clear how this methodology will be used in conjunction with a process planning system.

Vandenbrande [Van90] has developed a system that combines the principles of artificial intelligence and solid modeling. The program uses hints or clues to identify potential features in the boundary representation of a part (obtained from the PADL-2 solid modeler). The clues are generated by production rules and posted on a blackboard. The clues are assessed and the promising ones are pursued to recognize and extract the features. This system is capable of identifying interacting features (e.g. two crossing slots). This program also produces alternative feature interpretations in certain cases. Since there is no formalization available regarding the kinds of interactions it handles, it is hard to determine what all the interpretations it produces are.

2.5 Guidelines for Feature Extraction systems

Based on the merits and demerits on the existing feature extraction methodologies, the following can be stated as the desirable qualities of a feature extraction system for the machining domain: (These are based on the beliefs of the author, derived mostly from the existing literature.)

1. Features must be considered as volume elements, and at the same time a boundary representation of the features must be maintained (see Pratt [Pra88]). If features are modeled as

collections of surfaces, one runs into severe problems in dealing with feature interactions (see [Pra88, Min85] for a discussion on *face growing* and other problems).

2. Both topological and geometric information present in the BRep of a part should be used. A scheme based on topological information is not adequate. For example, consider the part shown in Figure 2.9. Based on topologically information alone, it is not possible to detect that the two steps are in the same plane, and hence can be considered as a single step. However, the topological information, being of a symbolic nature, should be preferred over the geometry information. The geometric information, being numeric in nature, is more prone to the problems of finite precision arithmetic. (See the book by Hoffmann [Hof89] for a discussion of the problems due to finite precision arithmetic in solid modeling and some solutions to this problem. See Weiler [Wei86]. for a discussion on the merits of the topological information.)

3. Feature interactions must be addressed and alternative feature interpretations of a part must be provided. Since this has been discussed earlier in this thesis, we will not elaborate further at this time.

4. The feature model must be integrated with the solid model and feature interactions must be inferred using the solid modeler. This is necessary to ensure that we have an unambiguous and unique representation of a part at all times. This requires that the solid modeler be able to handle arbitrary unions of solids, and hence the modeler must be of a non-manifold nature. This issue is described in greater detail by Pratt [Pra88].

5. The nominal geometry is only one of the many aspects in the description of a part. Various other aspects such as the tolerances, surface finish, material and the fixtures available, affect the choice of the features to represent a part, and hence advice from all these sources should be integrated in doing feature extraction. Smithers [Smi89] has stressed the importance of integrating multiple knowledge sources in CAD. Vandenbrande [Van90] discusses the role of integrating multiple knowledge sources in feature extraction.

The algebra of features that is described in subsequent chapters addresses all the issues mentioned above except the last.

Chapter 3

The Feature Algebra

Algebra is a cobra that makes my gunde gaabhara.
(Algebra is a cobra that makes my heart pound.)
—Telugu high school saying.

This chapter gives the mathematical treatment of the algebra of features that has been developed for this dissertation.

An algebraic structure [Pin82] in its simplest form is a set, with a rule (or rules) for combining its elements. Let A be any set. An *operation* $*$ on A is a rule which assigns to each ordered pair $\langle x, y \rangle$ of elements of A exactly one element $x * y$ in A. There are three aspects of the definition that need to be stressed :

1. $x * y$ is defined for every ordered pair, $\langle x, y \rangle$ of elements of A;

2. $x * y$ must be uniquely defined;

3. A is closed under the operation $*$.

In particular, a feature algebra is characterized by a set of features (denoted by $\mathcal{D}$), and binary operations on the features. Since these operations give meaningful values only for certain pairs of features, we include an element called INVALID in the set of features, to be used in cases where the operations do not produce meaningful values. By definition, for any operation $*$,

$$\forall x,\ x * \text{INVALID} = \text{INVALID} * x = \text{INVALID}$$

3.1 Mathematical Preliminaries

This section summarizes some of the mathematical concepts required for the definition of a feature in the next section. For a more detailed treatment of this material the reader is referred to Requicha and Tilove [RT78, Req77] and to Kuratowski [KM76], Mendelson [Men75], Simmons [Sim63] and Agoston [Ago76].

In the summary that follows, the symbols W and $\emptyset$ are used to denote the *universal set* and the empty set respectively. The symbols $\cup$, $\cap$, $-$ and c are used to denote the set operations union, intersection, difference and complement and the symbols $\subseteq$, $\supseteq$, $\subset$ and $\supset$ are used to denote relations subset, superset, proper subset and proper superset among sets.

3.1.1 Metric Spaces and Topological Spaces

If W is a non-empty set, a collection T of subsets of W is called a *topology* on W if it meets the following requirements:

1. $W \in T$ and $\emptyset \in T$.
2. The intersection of a finite number of sets of T belongs to T.
3. The union of a countable number of sets of T belongs to T.

An ordered pair (W, T) in which the first component W, is a non-empty set and the second component T is a topology on W is called a *topological space*. A subset of W is said to be *open* if and only if it belongs to T.

A *metric space* is an ordered pair (W, f), where W is a set, $\Re$ is the set of all reals, and $f : W \times W \rightarrow \Re$ is a function called the *distance* or *metric*, such that for all a, b, and c in W:

1. $f(a, b) \geq 0$;
2. $f(a, b) = 0$ if and only if $a = b$;
3. $f(a, b) = f(b, a)$;
4. $f(a, c) \leq f(a, b) + f(b, c)$ *(The Triangle Inequality).*

As a simple example, the ordered pair (E^3, d) where d is the function giving the Euclidean distance between two points, is a metric space.

Let (W, f) be a metric space, and x a point of W. The *open ball* of radius $R > 0$ about x, denoted by $Ball(x, R)$, is the set of all points y in W which satisfy $f(x, y) < R$.

Given a metric space (W, f), a set $X \subseteq W$ is *open* if it contains an open ball about each of its points. For example, given the metric space $M = (\Re, abs)$, where $abs(x, y) = \mid x - y \mid$, the set $(0, 10)$ of all reals greater than 0 and less than 10, is an open set.

Given a metric space (W, f), a set $X \subseteq W$ is *bounded* if it is a subset of a ball of finite radius.

Given a metric space (W, f), the set T of all (metric) open subsets of W form a topology on W. Thus, if (W, f) is a metric space, (W, T) is a topological space. Thus, the open sets of the metric space (W, f) and the topological space (W, T) are identical.

3.1.2 Closed Sets

A *neighborhood* of a point x in a topological space (W, T) is any subset of W which contains an open set which contains x.

Given a topological space (W, T) and a set $X \subseteq W$, a point x is a *limit point* of the set X if each neighborhood of x contains at least one point of X different from x. Notice that the limit points of a set need not belong to the set. Given the metric space $M = (\Re, abs)$ seen earlier, there is a corresponding topological space $N = (\Re, T)$ where T is the set of all (metric) open subsets of $\Re$. Now, consider the open interval $(0, 10)$. 0 and 10 are the limit points of the set $(0, 10)$ and neither of them belongs to the set.

The *closure* of a subset X, denoted by kX, is the union of X with the set of all its limit points.

A set X is *closed* if and only if $X = kX$. For example, the interval $[0, 10]$ of all reals greater than or equal to 0 and less than or equal to 10, is a closed set. Notice that some sets (the set $(0, 10]$, for example) are neither open nor closed.

3.1.3 Interior and Boundary

Given a topological space (W, T), a point x of W is an *interior point* of a set $X \subseteq W$ if X is a neighborhood of x, i.e., if X contains an open set which contains x.

Given a topological space (W, T), the *interior* of a set $X \subseteq W$, denoted by iX, is the set of all interior points of X.

$X = iX$ if and only if X is open.

Given a topological space (W, T), a point x of W is a *boundary point* of a set $X \subseteq W$ if each neighborhood of x intersects both X and cX.

The *boundary* of X, denoted by $b(X)$, is the set of all boundary points of X.

The reader may observe that the definitions of interior and boundary correspond to the intuitive notions of interior and boundary for solid objects.

3.1.4 Compactness

It can easily be seen that the ordered pair (E^n, f), where E^n is the Euclidean n-space and f is the *Euclidean distance*, is a metric space; and that there is a corresponding topological space (E^n, T) where T is the set of all open sets of E^n. Hence forth, we talk of the properties of a subset of E^n, where the corresponding metric and topological spaces are the ones just mentioned. A subset of Euclidean n-space is *compact* if and only if it is closed and bounded [Men75].

3.1.5 Regular Sets

The *regularization* of a subset X of W, denoted rX, is the set $rX = kiX$. A set X is *closed regular* if $X = rX$, i.e., if $X = kiX$. Note that $rrX = rX$. From now on, we simply refer to a closed regular set as a regular set. In intuitive terms, a regular set in E^3 cannot have any dangling faces, dangling edges or isolated points.

It is well known that when set-theoretic operations such as union, intersection, and difference are applied to two valid n-dimensional objects, the result is not necessarily a valid n-dimensional object. In particular, the intersection or difference of two regular n-dimensional objects need not be a regular n-dimensional object. For example, if two squares touch on one side, their intersection is a single line segment, which is not a valid two dimensional object. Requicha and Voelcker[RV85] have shown that this difficulty can be overcome by using regularized set operations instead of ordinary set operations. The symbols $\cup^*$, $\cap^*$, $-^*$ and c^* are used to denote regularized union, intersection, subtraction and complementation respectively. They are defined below:

$$X \cup^* Y = r(X \cup Y)$$

$$X \cap^* Y = r(X \cap Y)$$

$$X -^* Y = r(X - Y)$$

$$c^* X = rcX$$

Figure 3.1 shows the difference between ordinary and regularized set operations.

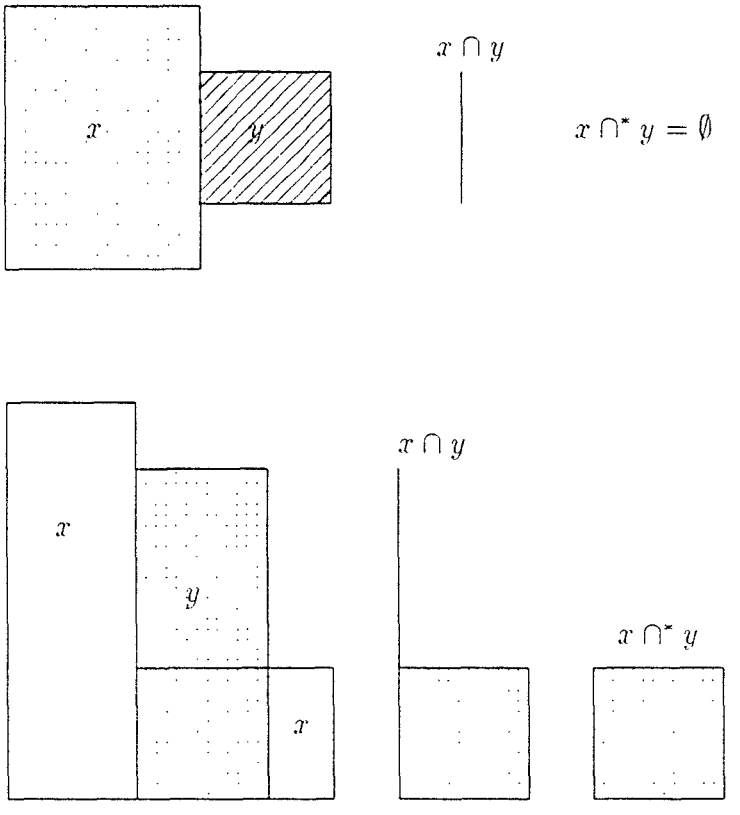


Figure 3.1: Ordinary and Regularized set operations.

3.1.6 Semi-Analytic Sets

A function $f : E^3 \rightarrow \Re$ is said to be *analytic* [Ful69] throughout its domain if it can be expanded in a power series in x, y and z about every point in its domain. A necessary condition for a (real) function to be analytic is that it be infinitely differentiable. A subset of E^3 is said to be *semi-analytic* if it is a finite combination, via the set operations union, intersection and complement, of sets X_i of the form

$$X_i = \{p \in E^3 : f_i(p) \geq 0\},$$

where f_i is any analytic function on E^3.

It can be shown that the interior, boundary, and closure of a semi-analytic set is also semi-analytic, and that class of compact, regular, semi-analytic sets is closed under regularized set operations (regularized union, intersection and difference.)

3.1.7 Convexity and Concavity

Given two distinct points p_1 and p_2 in Euclidean n-space, the *convex combination* of p_1 and p_2 is the set

$$\{p : p = \alpha p_1 + (1 - \alpha)p_2, \alpha \in \Re, 0 \leq \alpha \leq 1\}.$$

The convex combination normally describes the *straight line segment* $\overline{p_1 p_2}$ (unordered pair). A subset x of E^3 is said to be *convex* if and only if for any two points p_1 and p_2 belonging to x, the segment $\overline{p_1 p_2}$ is entirely contained in x. In this thesis, a subset of E^3 that is not convex is said to be *concave*.

It can be shown that the intersection of two convex sets is a convex set [PS85]. It can easily be seen that the union and difference of two convex sets is not necessarily a convex set.

3.2 Feature Definition

Thus far, we have seen the definitions of compact, regular and semi-analytic sets which are well-known mathematical and topological concepts. In the remainder of the chapter, we will discuss the feature algebra, which is the primary contribution of this thesis. The concepts discussed earlier will be useful in the definition of features.

A feature (other than INVALID) is given by a pair $x = \langle \text{ps}, \text{patches} \rangle$ where the two entries in the pair satisfy the following conditions:

1. The first entry ($\text{ps}(x)$) is the set of all points in a feature and is a subset of E^3 that is compact, regular and semi-analytic.

2. The second entry ($\text{patches}(x)$) is a partition of the boundary of $\text{ps}(x)$ into labeled patches (as defined below).

Henceforth, the boundary of $\text{ps}(x)$ and the patches in $\text{patches}(x)$ will be referred to as the boundary and patches of x.

It is worthwhile now to examine the scope and significance of the above definition. Regularity restricts a feature to be homogeneously three dimensional. In other words, a feature may not have dangling faces, edges and vertices. Figure 3.2 shows examples of valid and invalid features. Even parts with sheet metal components have a finite thickness, so this appears to be a very reasonable restriction. Earlier we noted that a subset of E^n is compact if and only if it is closed and bounded. The set of points of a feature constitute a regular set. Hence, they are closed [RT78]. Compactness,

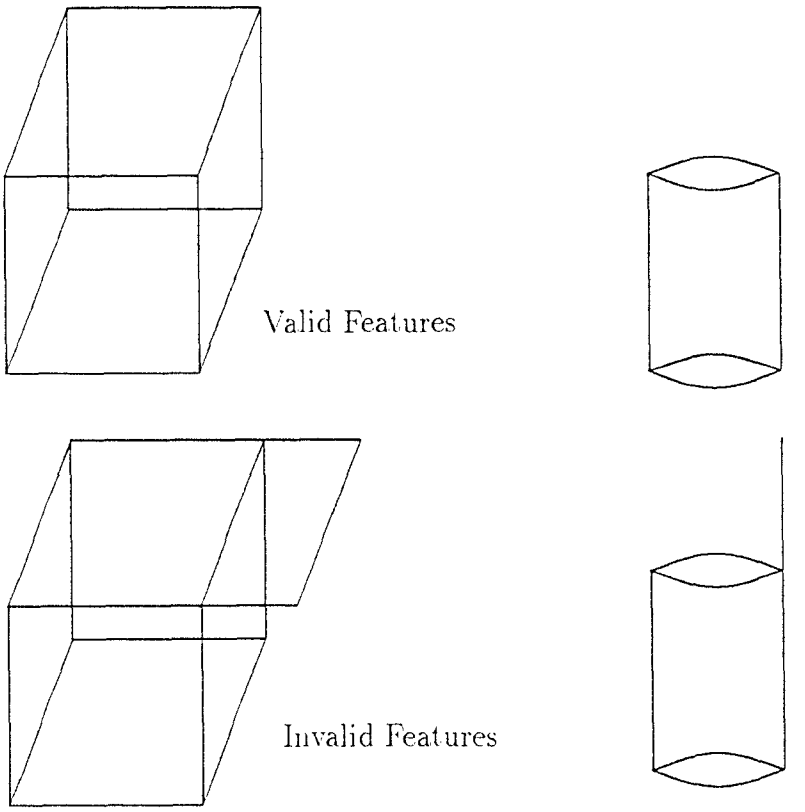


Figure 3.2: Valid and Invalid Features.

therefore restricts a feature to be bounded which is met by all features of finite dimensions. The domain of semi-analytic sets covers practically all the shapes of interest to manufacturing. The reader may note that all planar polyhedra, cylinders, cones, spheres, tori and a variety of sculptured surfaces are encompassed by this set. It also includes concave features such as T-slots, counter bores and countersinks (see Figure 3.3).

A *patch* is a regular, semi-analytic subset of the boundary of a feature. Figure 3.4 illustrates some examples of patches. A labeled patch is a patch p with a label label(p) whose value is either BLOCKED or UNBLOCKED. The patches and their labels are intended to describe what the boundaries of the feature mean in the real world. In particular, suppose x is a feature of some manufacturable object X. If some patch p of x separates metal from air (i.e., p lies on the boundary of X), then p is BLOCKED; and if p separates air from air (i.e., p does not lie on the boundary of

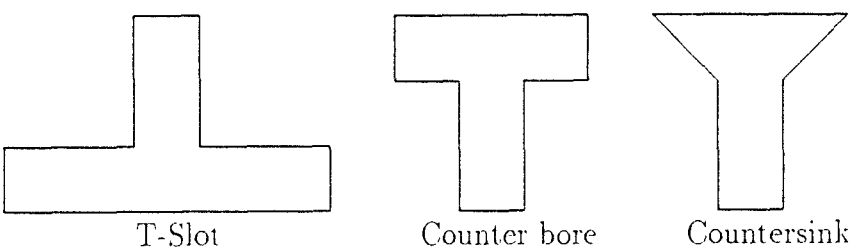


Figure 3.3: Cross sectional views of three concave features.

X), then p is UNBLOCKED. If a patch cannot we labeled as BLOCKED or UNBLOCKED , then we partition the patch into smaller patches, each of which can be labeled consistently. Figure 3.5 shows the BLOCKED and UNBLOCKED patches for the example discussed in Figure 1.1. The information regarding BLOCKED and UNBLOCKED patches is useful in process planning. In particular, if a patch is labeled UNBLOCKED, it is optional to machine the patch and if patch is labeled BLOCKED, it is necessary to machine the patch. Given a feature x and a patch p of x, the regularized complement of p is defined as $c^*(p,x) = b(\mathrm{ps}(x)) -^* p$. From the definition, it follows that the boundary of a feature is a patch. From the properties of the compact, regular and semi-analytic sets stated earlier, one can see that the regularized complement of a patch is also a patch.

Proposition 1 *The set of all patches of a feature is closed under regularized union, intersection, complement and difference.*

The above proposition is a direct consequence of the closure properties of compact, regular, semi-analytic sets stated earlier.

3.2.1 Neighborhood

The notion of a neighborhood of a point on the boundary of a feature will be useful later on. If $\delta > 0$ the δ-neighborhood $N(p,\delta)$ of a point p on the boundary of a feature x, is the set of all points on the boundary of x that are at a distance $\leq \delta$ from p.

3.2.2 Orthogonal Projection

In this section a function known as *orthogonal projection* will be defined. It will be used later in describing the operations in the algebra.

Given a planar patch s, and a point p not in the plane of s, the orthogonal projection of p in s, denoted $\mathcal{O}(p,s)$ is the point p' (if it exists) on s such that the line through p and p' is perpendicular to the plane of s.

Given two planar patches s_1 and s_2, the orthogonal projection of s_1 in s_2 (if it exists), denoted $\mathcal{O}(s_1,s_2) = \{\mathcal{O}(p,s_2) \mid p \in s_1\}$, if $\mathcal{O}(p,s_2)$ is defined for all $p \in s_1$.

3.3 Operations on Features

This section describes the operations on features. In order to describe them, one needs to first understand the methodology for classifying the patches of the boundary of one solid with respect to another solid. This methodology has been developed by Vaněček[Van89b] who used it in the context of performing set operations on planar polyhedra.

3.3.1 Patch Classification Scheme

A patch x_i of the boundary of a solid x is *homogeneous* with respect to solid y if one or more of the classification relationships holds:

1. x_i **IN** y; i.e., the interior of x_i lies in the interior of y.
2. x_i **OUT** y; i.e., the interior of x_i is outside of y.

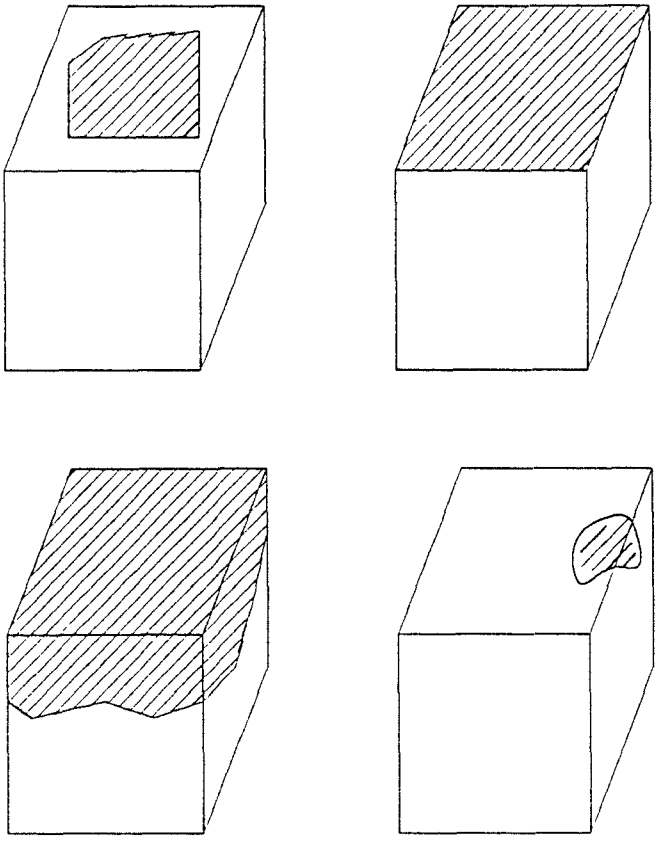


Figure 3.4: Some examples of patches (shaded).

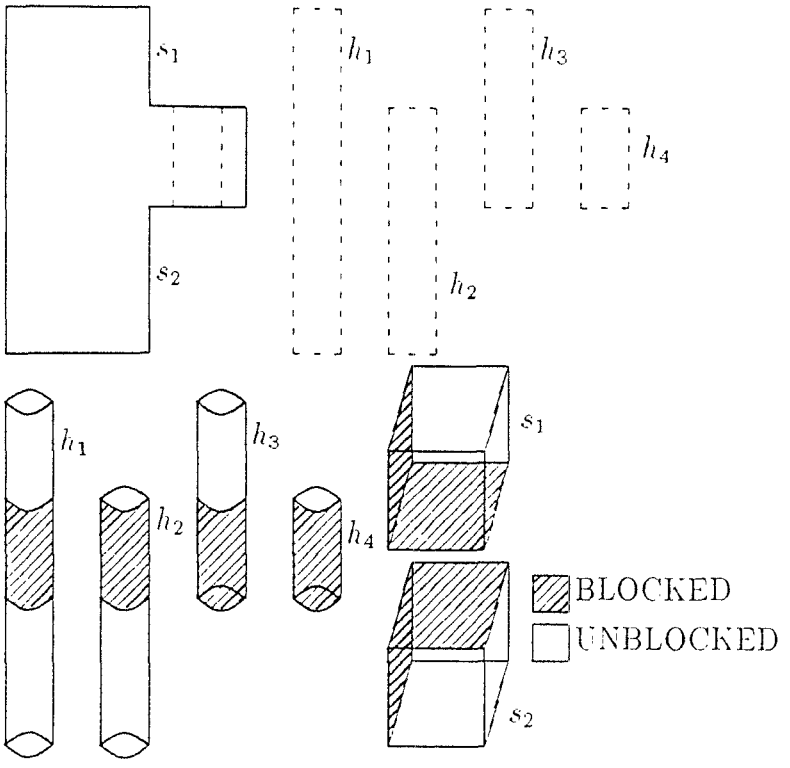


Figure 3.5: Blocked and Unblocked patches of features.

3. x_i **WITH** y; i.e., x_i lies on the boundary of y, and both x and y are on the same side of the boundary.

4. x_i **ANTI** y: i.e., x_i lies on the boundary of y, and x and y are on the opposite sides of the boundary.

Figure 3.6 illustrates the patch classification scheme described above. For example, the patch (also face) v_1 of the solid v is not homogeneous with respect to w. However, it can be partitioned into two patches each of which is homogeneous with respect to w.

Given the above patch classification scheme, the boundary of a solid x can be partitioned into a set of patches ($\mathcal{P}$), such that each patch in $\mathcal{P}$ is homogeneous with respect to y. Thus, one can obtain collections of patches $x\mathbf{IN}y$, $x\mathbf{OUT}y$, $x\mathbf{WITH}y$ and $x\mathbf{ANTI}y$ given by the following definitions:

$$\begin{aligned} x\mathbf{IN}y &= \{p \in \mathcal{P} | p\mathbf{IN}y\} \\ x\mathbf{OUT}y &= \{p \in \mathcal{P} | p\mathbf{OUT}y\} \\ x\mathbf{WITH}y &= \{p \in \mathcal{P} | p\mathbf{WITH}y\} \\ x\mathbf{ANTI}y &= \{p \in \mathcal{P} | p\mathbf{ANTI}y\}. \end{aligned}$$

The operations in the feature algebra are defined in terms of set operations on solids. Using the above classification sets, the boundaries of $x \cup^* y$, $x \cap^* y$, $x -^* y$ and $y -^* x$ can be computed as given below. For any patch p, p^{-1} is the same as p with the sign of the normal to the patch at every point reversed.

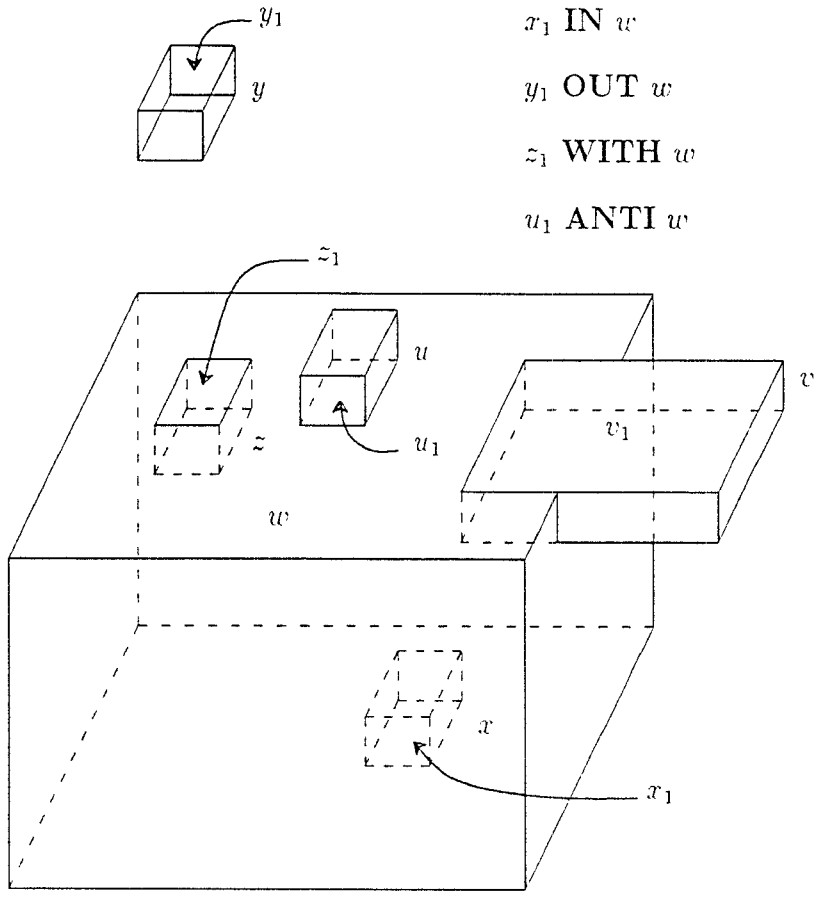


Figure 3.6: The classification of patches of solids x, y, z, u and v with respect to the solid w

$$
\begin{aligned}
b(x \cup^* y) &= x\mathbf{OUT}y \cup y\mathbf{OUT}x \cup x\mathbf{WITH}y \\
b(x \cap^* y) &= x\mathbf{IN}y \cup y\mathbf{IN}x \cup y\mathbf{WITH}x \\
b(x -^* y) &= x\mathbf{OUT}y \cup (y\mathbf{IN}x)^{-1} \cup x\mathbf{ANTI}y \\
b(y -^* x) &= y\mathbf{OUT}x \cup (x\mathbf{IN}y)^{-1} \cup y\mathbf{ANTI}x.
\end{aligned}
$$

3.3.2 Truncation

Given two features x and y, the truncation operation ($\mathcal{T}$) is defined as follows: $z = x\mathcal{T}y = \langle u, v \rangle$, where $u = \text{ps}(x) -^* \text{ps}(y)$ and v is a collection of patches satisfying the following properties:

1. the union of all the patches in v is $b(u)$.
2. every patch in v that belongs to $\text{ps}(x)\mathbf{OUT}\text{ps}(y)$ or $\text{ps}(x)\mathbf{ANTI}\text{ps}(y)$ is a subset of some patch of x.

Note that,

$$
b(\text{ps}(x) -^* \text{ps}(y)) = \text{ps}(x)\mathbf{OUT}\text{ps}(y) \cup (\text{ps}(y)\mathbf{IN}\text{ps}(x))^{-1} \cup \text{ps}(x)\mathbf{ANTI}\text{ps}(y)
$$

The labels of the patches of v are determined as follows:

For every patch p of v, if $p \in (\text{ps}(x)\mathbf{OUT}\text{ps}(y))$ or $p \in (\text{ps}(x)\mathbf{ANTI}\text{ps}(y))$, let p_1 be the patch of x such that $p \subseteq p_1$.

$$
\text{label}(p) = \begin{cases} \text{label}(p_1) & \text{if } p \in \text{ps}(x)\mathbf{OUT}\text{ps}(y) \text{ or } p \in \text{ps}(x)\mathbf{ANTI}\text{ps}(y) \\ \text{UNBLOCKED} & \text{otherwise} \end{cases} \tag{3.1}
$$

The rationale for the above labeling is as follows: If a patch p of $x\mathcal{T}y$ belongs to $\text{ps}(x)\mathbf{OUT}\text{ps}(y)$ $\cup$ $\text{ps}(x)\mathbf{ANTI}\text{ps}(y)$, then $p \subset b(\text{ps}(x))$ and hence, the patch p must have the same label as a patch p_1 of x that contains p. If there is no single patch p_1 that contains p then we can either split p or combine some patches of x to satisfy this property. If a patch p of $x\mathcal{T}y$ belongs to $(\text{ps}(y)\mathbf{IN}\text{ps}(x))^{-1}$, then this patch is **IN** with respect to $\text{ps}(x)$, and hence the interior of the patch p cannot belong to the boundary of the final part. Therefore, p should be labeled UNBLOCKED. Figure 3.7 illustrates the truncation operation through some examples.

3.3.3 Infinite Extension

In this section the *infinite extension* of a feature with respect to a patch will be defined. This is used subsequently in defining an operation called *maximal extension*.

Let us consider a patch p on a feature x.

At any point $p_i = \langle x_i, y_i, z_i \rangle$ on the boundary of p (b(p)), consider the neighborhood $N(p_i, \delta_i)$ for some arbitrarily small $\delta_i > 0$. Suppose there exists an analytic function f such that for every point $p' \in N(p_i, \delta_i) - p$, $f(p') = 0$ and the following limits exist:

$$
\lim_{\delta_i \to 0} f_x(p') = f_{ix}
$$

$$
\lim_{\delta_i \to 0} f_y(p') = f_{iy}
$$

$$
\lim_{\delta_i \to 0} f_z(p') = f_{iz}
$$

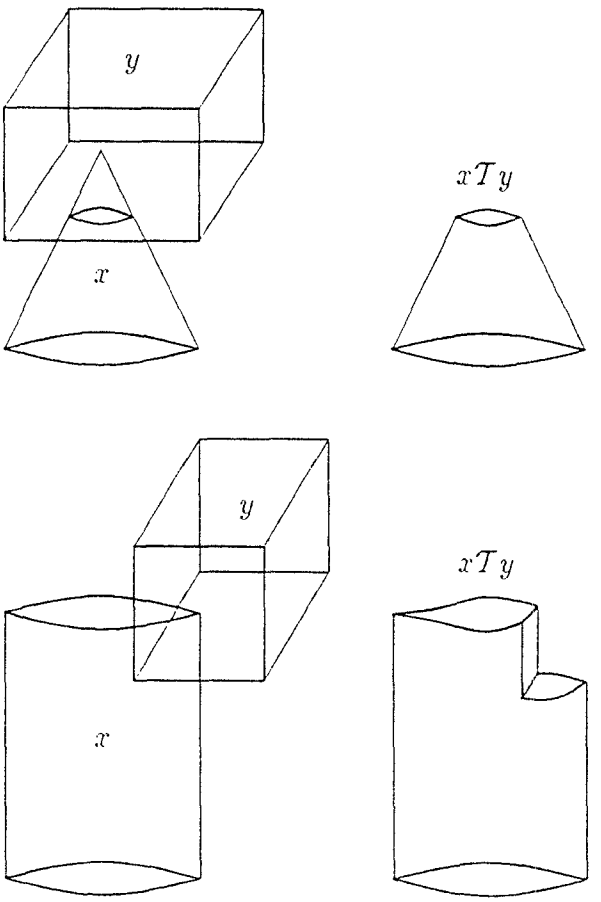


Figure 3.7: Examples of the truncation operation.

where $f_x(p')$, $f_y(p')$ and $f_z(p')$ are the partial derivatives of f with respect to X, Y and Z.

Now, we define what is called the *partial-tangent-plane* $\bar{\mathcal{T}}_p(p_i, x) = 0$ as follows:

$$\bar{\mathcal{T}}_p(p_i, x) = \begin{cases} f_{ix}(X - x_i) + f_{iy}(Y - y_i) + f_{iz}(Z - z_i) & \text{if } f_{ix},\ f_{iy} \text{ and } f_{iz} \text{ exist} \\ \text{INVALID} & \text{otherwise} \end{cases}$$

$\bar{\mathcal{T}}_p(p_i, x) = 0$ divides E^3 into two planar half-spaces and the one containing the inward-pointing normal to the feature x at p_i is denoted by $\mathcal{G}_p(p_i, x)$.

Now, we define the function $\mathcal{C}_p$ as follows:

$$\mathcal{C}_p(x) = \bigcap_{p_i} \mathcal{G}_p(p_i, x)\ \forall p_i \in b(p) \text{ wherever } \mathcal{G}_p(p_i, x) \text{ is defined}$$

At any point p_i on p let us denote the tangent plane by $\mathcal{T}_p(p_i, x) = 0$. $\mathcal{T}_p(p_i, x) = 0$ divides E^3 into two planar half-spaces and the one containing the inward-pointing normal to the feature x at p_i is denoted by $\mathcal{H}_p(p_i, x)$.

Now, we define the function $\mathcal{A}_p$ as follows:

$$\mathcal{A}_p(x) = \bigcap_{p_i} \mathcal{H}_p(p_i, x)\ \forall p_i \in p \text{ wherever } \mathcal{H}_p(p_i, x) \text{ is defined}$$

Let us denote $c^*\mathcal{A}_p(x)$ by $\mathcal{B}_p(x)$

Now, $\mathcal{I}_p(x)$ is defined as:

$$\mathcal{I}_p(x) = (\mathcal{B}_p(x) \cap \mathcal{C}_p(x)) \cup \text{ps}(x)$$

Now, we will illustrate the notion of infinite extension through some examples. Consider the feature f showed in Figure 3.8. Figure 3.8 (a) shows the patch p that has been identified on the feature. To compute $\mathcal{C}_p(f)$, we first construct tangent planes on the boundary of the patch and consider the intersection of the half-spaces. The $\mathcal{C}_p(f)$ thus obtained in shown in Figure 3.8 (b). To compute $\mathcal{A}_p(f)$, we construct the tangent planes at each point on the patch p. The resulting $\mathcal{A}_p(f)$ is shown in Figure 3.8 (c). $\mathcal{B}_p(f)$, which is the regularized complement of $\mathcal{A}_p(f)$ is shown in Figure 3.8 (d). Finally, $\mathcal{I}_p(f)$ is shown in Figure 3.8 (e).

Figures 3.9 (a) through (e) and Figures 3.10 (a) through (e) illustrate the same ideas for a second and a third example.

3.3.4 Maximal Extension

Given two features x and y, and a patch p on x, the maximal extension of x in y with respect to a patch p (denoted by $x\mathcal{M}_p y$) is defined as follows:

$$x\mathcal{M}_p y = \langle u, v \rangle$$

where $u = \mathcal{I}_p(x) \cap^* (\text{ps}(x) \cup^* \text{ps}(y))$ and v is a collection of patches satisfying the following properties:

1. the union of all patches in v is $b(u)$.
2. every patch in v that belongs to $(\text{ps}(x) \cup^* \text{ps}(y))\mathbf{IN}\mathcal{I}_p(x)$ or $(\text{ps}(x) \cup^* \text{ps}(y))\mathbf{WITH}\mathcal{I}_p(x)$ is a subset of some patch of $\text{ps}(x) \cup^* \text{ps}(y)$.

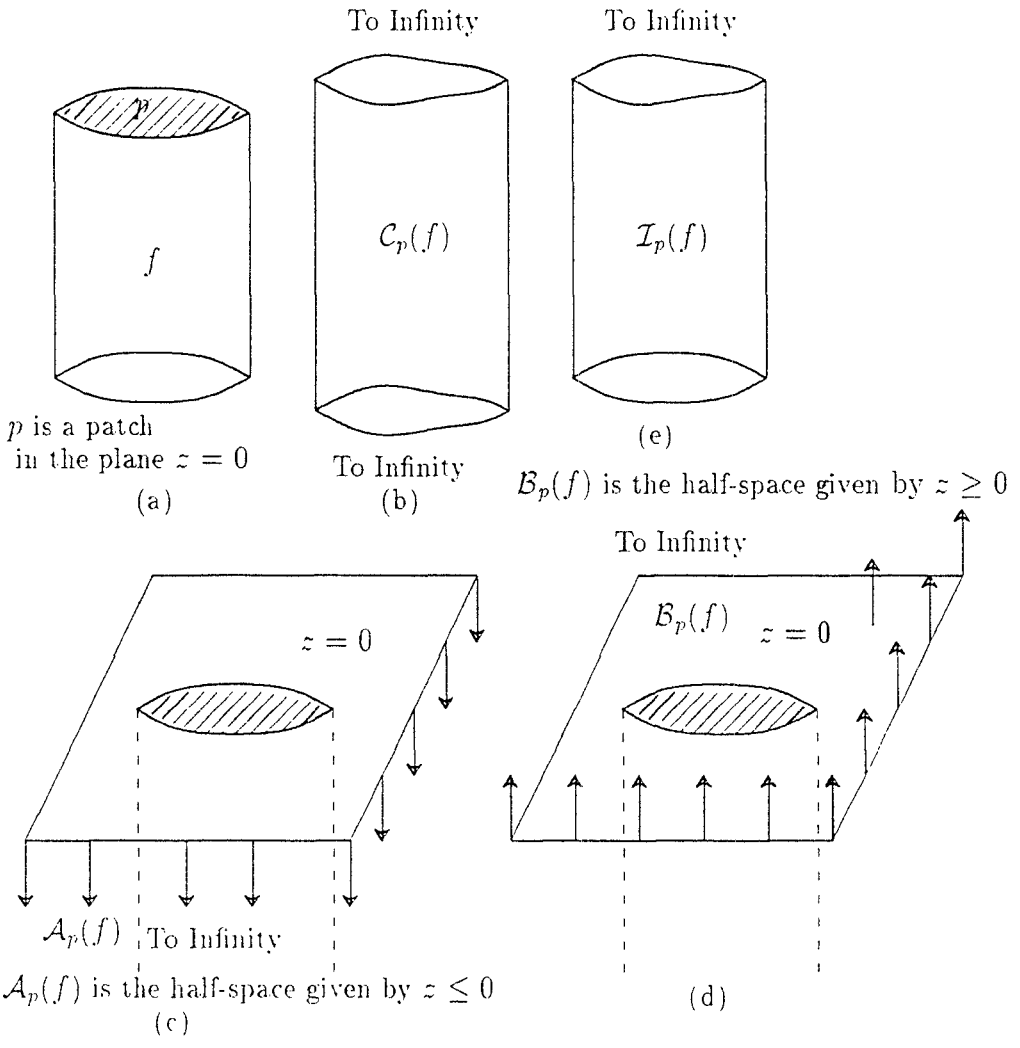


Figure 3.8: Example illustrating Infinite Extension.

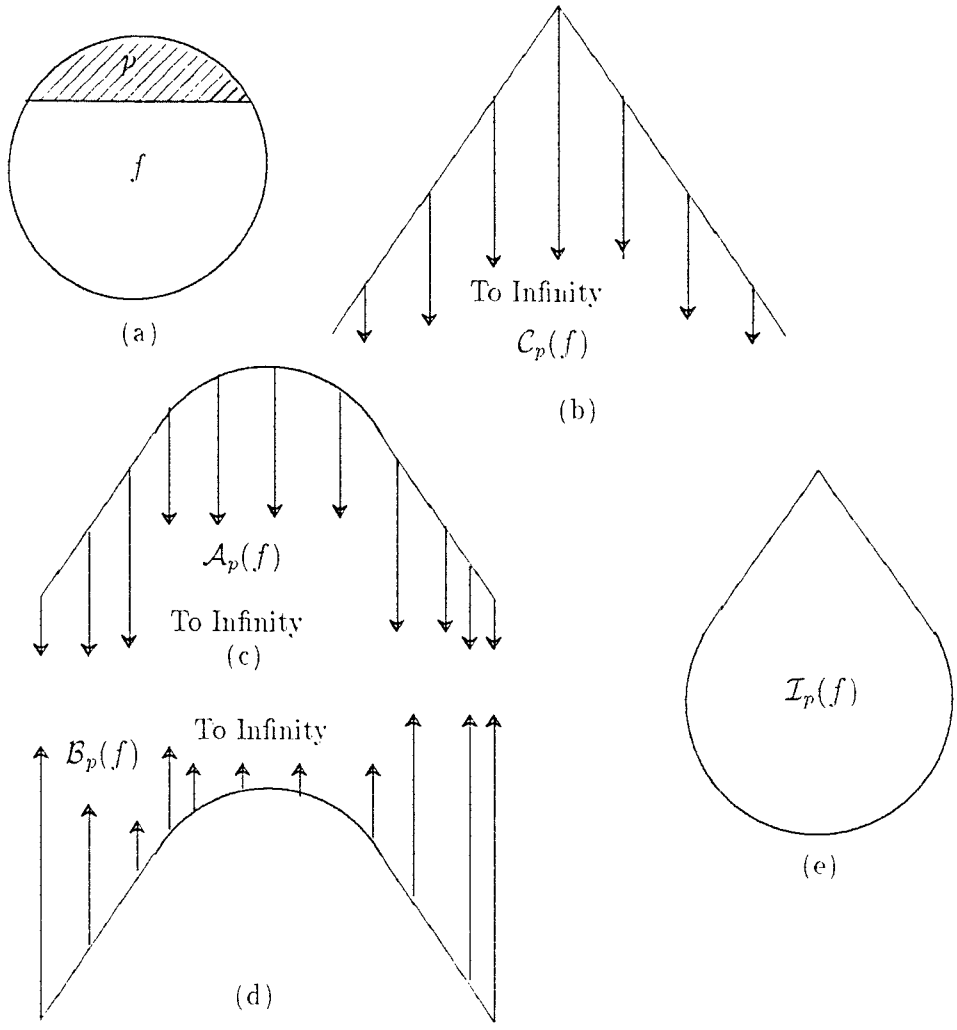


Figure 3.9: Example illustrating Infinite Extension.

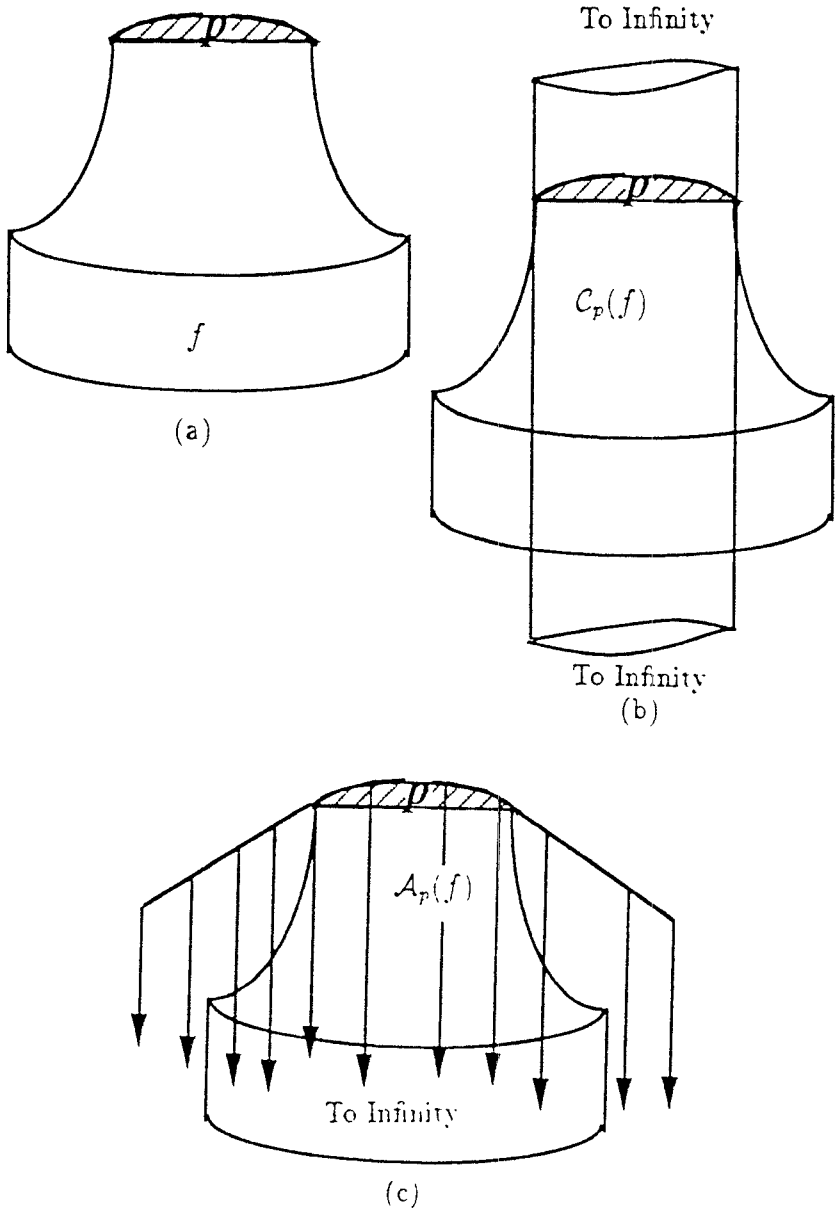


Figure 3.10: Example illustrating Infinite Extension.

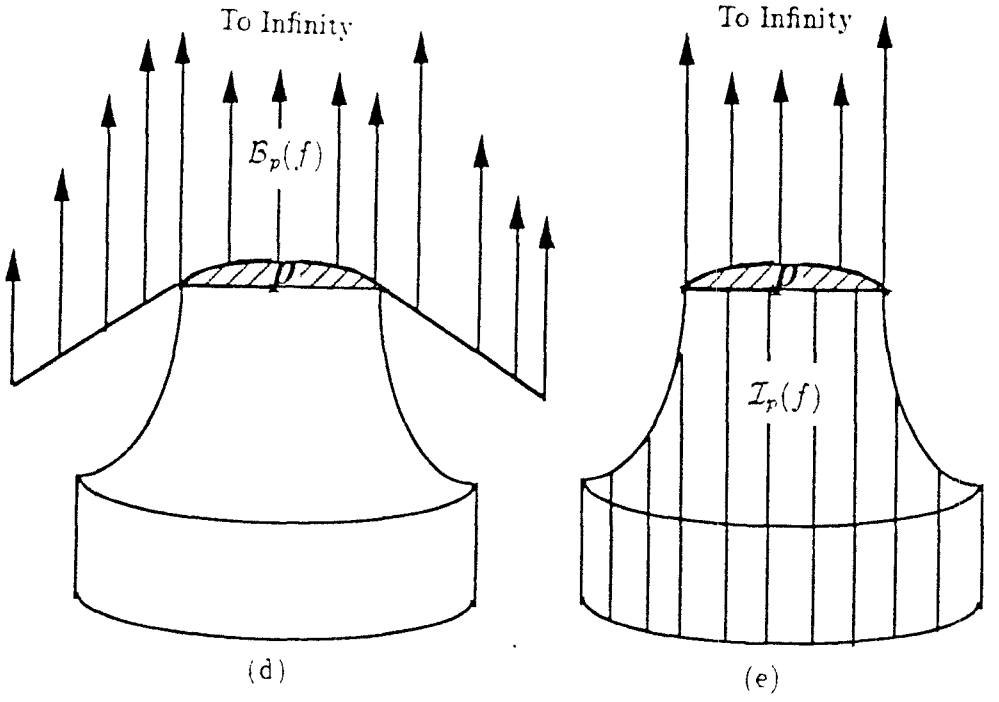


Figure 3.10 Example illustrating Infinite Extension.

Note that,

$$\begin{aligned} b(u) &= \mathcal{I}_p(x)\mathbf{IN}(\mathrm{ps}(x) \cup^* \mathrm{ps}(y)) \cup (\mathrm{ps}(x) \cup^* \mathrm{ps}(y))\mathbf{IN}\mathcal{I}_p(x) \cup \\ &\quad (\mathrm{ps}(x) \cup^* \mathrm{ps}(y))\mathbf{WITH}\mathcal{I}_p(x) \end{aligned}$$

The labels of the patches of v are determined as follows: For every patch p of v, if $p \in (\mathrm{ps}(x) \cup^* \mathrm{ps}(y))\mathbf{IN}\mathcal{I}_p(x)$ or $p \in (\mathrm{ps}(x) \cup^* \mathrm{ps}(y))\mathbf{WITH}\mathcal{I}_p(x)$ let p_1 be the patch of $\mathrm{ps}(x) \cup^* \mathrm{ps}(y)$ such that $p \subseteq p_1$.

$$\mathrm{label}(p) = \begin{cases} \mathrm{label}(p_1) & \text{if } p \in (\mathrm{ps}(x) \cup^* \mathrm{ps}(y))\mathbf{IN}\mathcal{I}_p(x) \text{ or} \\ & p \in (\mathrm{ps}(x) \cup^* \mathrm{ps}(y))\mathbf{WITH}\mathcal{I}_p(x) \\ \mathrm{UNBLOCKED} & \text{otherwise} \end{cases}$$

The rationale for the above scheme of labeling the patches is as follows: If a patch p of $x\mathcal{M}_p y$ belongs to $(\mathrm{ps}(x) \cup^* \mathrm{ps}(y))\mathbf{IN}\mathcal{I}_p(x) \cup (\mathrm{ps}(x) \cup^* \mathrm{ps}(y))\mathbf{WITH}\mathcal{I}_p(x)$, then $p \subset b(\mathrm{ps}(x) \cup^* \mathrm{ps}(y))$ and hence, the patch p must have the same label as a patch p_1 of $\mathrm{ps}(x) \cup^* \mathrm{ps}(y)$ that contains p. If there is no single patch p_1 that contains p then we can either split p or combine some patches of $b(\mathrm{ps}(x) \cup^* \mathrm{ps}(y))$ to satisfy this property. If a patch p of $x\mathcal{M}_p y$ belongs to $\mathcal{I}_p(x)\mathbf{IN}(\mathrm{ps}(x) \cup^* \mathrm{ps}(y))$, then this patch is **IN** with respect to $\mathrm{ps}(x) \cup^* \mathrm{ps}(y)$, and hence the interior of the patch p cannot belong to the boundary of the final part. Therefore, p should be labeled UNBLOCKED.

For the definition to be complete, the labels of the patches of $\mathrm{ps}(x) \cup^* \mathrm{ps}(y)$ must be defined, given the labels of the patches of x and y. As stated earlier,

$$b(\mathrm{ps}(x) \cup^* \mathrm{ps}(y)) = \mathrm{ps}(x)\mathbf{OUT}\mathrm{ps}(y) \cup \mathrm{ps}(y)\mathbf{OUT}\mathrm{ps}(x) \cup \mathrm{ps}(x)\mathbf{WITH}\mathrm{ps}(y)$$

Let us denote a arbitrary patch of $b(\mathrm{ps}(x) \cup^* \mathrm{ps}(y))$ by p. If p is a patch of $\mathrm{ps}(x)\mathbf{OUT}\mathrm{ps}(y)$ or $\mathrm{ps}(x)\mathbf{WITH}\mathrm{ps}(y)$, let p_1 be a patch of x such that $p \subseteq p_1$. If p is a patch of $\mathrm{ps}(y)\mathbf{OUT}\mathrm{ps}(x)$ or $\mathrm{ps}(x)\mathbf{WITH}\mathrm{ps}(y)$, let p_2 be the patch of y such that $p \subseteq p_2$.

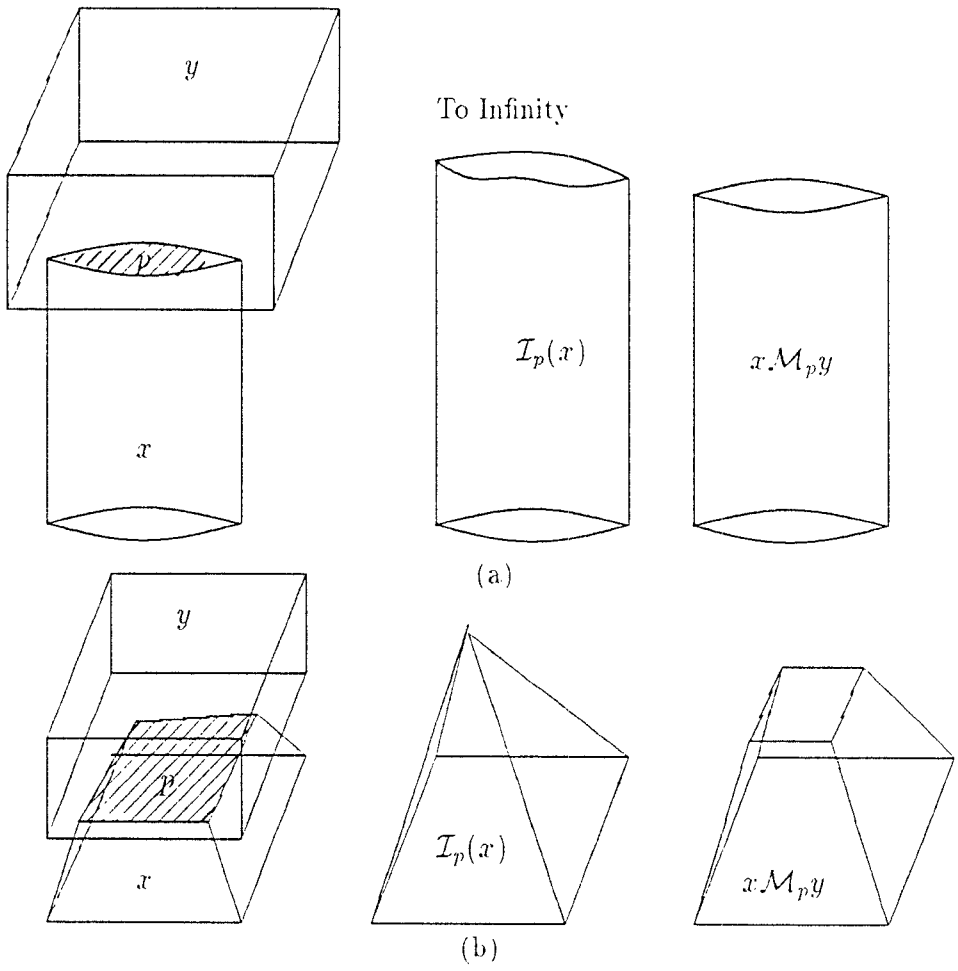


Figure 3.11: Examples of maximal extension operation.

The labels of the patches of $b(\text{ps}(x) \cup^* \text{ps}(y))$ are defined below:

$$\text{label}(p) = \begin{cases} \text{label}(p_1) & \text{if } p \in \text{ps}(x)\mathbf{OUT}\text{ps}(y) \\ \text{label}(p_2) & \text{if } p \in \text{ps}(y)\mathbf{OUT}\text{ps}(x) \\ \text{label}(p_1) & \text{if } p \in \text{ps}(x)\mathbf{WITH}\text{ps}(y) \end{cases}$$

The rationale for the above labeling scheme is as follows: If a patch p of $\text{ps}(x) \cup^* \text{ps}(y)$ belongs to $\text{ps}(x)\mathbf{OUT}\text{ps}(y)$, then $p \subset b(\text{ps}(x))$ and hence, the patch p must have the same label as a patch p_1 of x that contains p. If a patch p of $\text{ps}(x) \cup^* \text{ps}(y)$ belongs to $\text{ps}(y)\mathbf{OUT}\text{ps}(x)$, then $p \subset b(\text{ps}(y))$ and hence, the patch p must have the same label as a patch p_2 of y that contains p. If a patch p of $\text{ps}(x) \cup^* \text{ps}(y)$ belongs to $\text{ps}(x)\mathbf{WITH}\text{ps}(y)$, then $p \subset b(\text{ps}(x))$ and $p \subset b(\text{ps}(y))$ and hence, the patch p must have the same label as a patch p_1 of x that contains p or the patch p_2 of y that contains p. (Both p_1 and p_2 will have the same label). In the above cases, if there is no single patch p_1 that contains p then we can either split p or combine some patches of $b(\text{ps}(x))$ or $b(\text{ps}(y))$ to satisfy the above properties. Figure 3.11 (a) and Figure 3.11 (b) illustrate some examples of maximal extension.

3.3.5 Second Infinite Extension

In this section, we define a second kind of infinite extension known as the *second infinite extension.* The second infinite extension of a feature x with respect to a patch p is denoted by $\mathcal{I}_p^2(x)$. Using the second infinite extension, the *second maximal extension* $\mathcal{M}_p^2$ operation can be defined by substituting $\mathcal{I}_p^2(x)$ for $\mathcal{I}_p(x)$ in the definition of *maximal extension.*

Let x be any feature and p_i be any point on x for which there exists a plane tangent to x at p_i. Then $H(p_i, x)$ is the closed half-space tangent to x at p_i that contains x.

The second infinite extension of x with respect to the patch p is

$$\mathcal{I}_p^2(x) = \bigcap \{H(p_i, x) | p_i \in c^*(p, x).\}$$

Figure 3.12 shows a countersink feature x, two patches a and b of x, and the corresponding $\mathcal{I}_a^2(x)$, $\mathcal{I}_b^2(x)$, $x\mathcal{M}_a^2 x$ and $x\mathcal{M}_b^2 x$. In this figure, a is the union of the cylindrical portion of the countersink and the bottom face; and b is the union of the conical portion of the countersink and the top face. One may note that $a \cup b = b(\text{ps}(x))$.

The following observations may be made regarding second infinite extension and second maximal extension:

1. In some cases, $\mathcal{I}_p(x)$ may be equal to $\mathcal{I}_p^2(x)$. In particular, for rectangular solids and cylinders, if we choose the planar faces of the feature to be a patch we find that $\mathcal{I}_p(x) = \mathcal{I}_p^2(x)$. We will find this observation useful in Chapter 4. It may be noted that $\mathcal{I}_p(x) = \mathcal{I}_p^2(x)$ for the two examples given in Figures 3.8, 3.9 and 3.10.

2. $\mathcal{I}_p$ satisfies a useful property viz., $\text{ps}(x) \subseteq \mathcal{I}_p(x)$ (see Proposition 6). As one can easily verify, this is not true of $\mathcal{I}_p^2$ (see Figure 3.12). This makes $\mathcal{I}_p^2$ less useful, and we will discuss why, in Section 3.4.

3. If $\text{ps}(x)$ is a convex set, then for all patches p of x, $\text{ps}(x) \subseteq \mathcal{I}_p(x)$.

There are two reasons to study second infinite extension. First, since $\mathcal{I}_p^2(x) = \mathcal{I}_p(x)$ in a lot of cases, it is simpler to use the definition of $\mathcal{I}_p^2(x)$. Second, for concave features such as the countersink, we need to use the definition of $\mathcal{I}_p^2(x)$. Whether or not $\mathcal{I}_p = \mathcal{I}_p^2$ for all convex features is yet to be proved.

3.3.6 Combination

Given two features x and y, the combination operation ($\mathcal{C}$) is defined as follows: $z = x\mathcal{C}y = \langle u, v \rangle$, where $u = \text{ps}(x) \cup^* \text{ps}(y)$ and v is a collection of patches satisfying the following properties:

1. the union of all the patches in v is $b(u)$.

2. every patch in v that belongs to $\text{ps}(x)\mathbf{OUT}\text{ps}(y)$ or $\text{ps}(x)\mathbf{WITH}\text{ps}(y)$ is a subset of some patch of x.

3. every patch in v that belongs to $\text{ps}(y)\mathbf{OUT}\text{ps}(x)$ or $\text{ps}(x)\mathbf{WITH}\text{ps}(y)$ is a subset of some patch of y.

Note that,

$$b(\text{ps}(x) \cup^* \text{ps}(y)) = \text{ps}(x)\mathbf{OUT}\text{ps}(y) \cup \text{ps}(y)\mathbf{OUT}\text{ps}(x) \cup \text{ps}(x)\mathbf{WITH}\text{ps}(y)$$

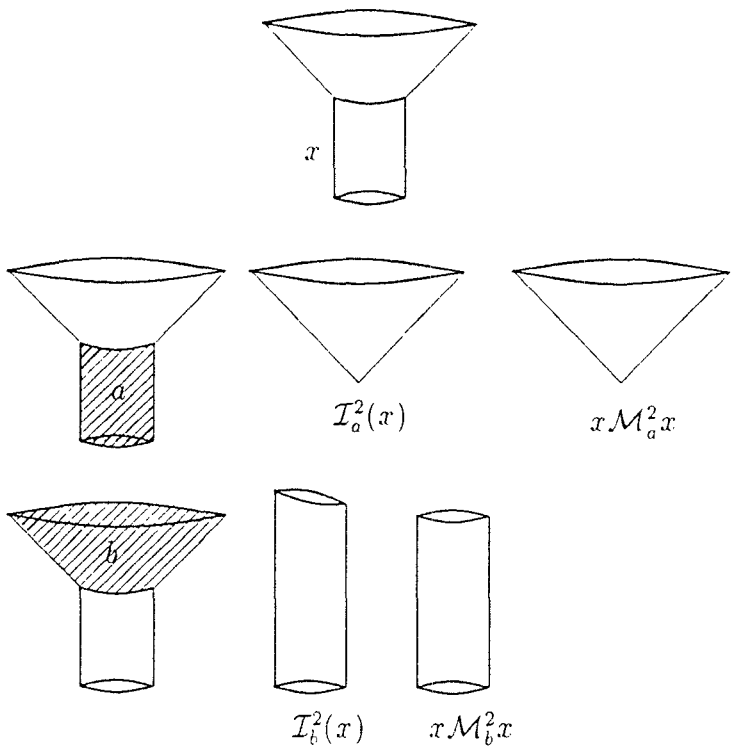


Figure 3.12: Example illustrating the second infinite extension and the second maximal extension.

The patch label propagation for this case has already been discussed in Section 3.3.4. An example of a case where the $\mathcal{C}$ operation is useful is a situation where two (or more) pockets can be combined into a single pocket with a complex contour. Figure 3.13 shows a part with two rectangular pockets x and y. They are combined into a single contoured pocket using the $\mathcal{C}$ operation. If we have only rectangular pockets in our domain, we can obtain the result of the $\mathcal{C}$ operation with the $\mathcal{M}_p$ operation with an appropriately chosen patch.

3.4 Properties of the Algebra of Features

In this section, certain properties of the algebra of features are discussed. The goal is to generate new features from the features one already has, using the operations discussed earlier. During this process, one would not like to generate a feature anew if it can be shown that a feature with the same set of points has already been generated. To compute new features from existing ones, one must perform set operations on solids. In most instances arising in practice, the existing algorithms for performing set operations have an average case complexity of $O(n \log n)$, where n is the number of topological entities in a solid (such as vertices, edges and faces). Thus, set operations on solids are expensive computations and should be minimized or substituted by cheaper operations. Later on, when we discuss the implementation of a restricted feature algebra, we will describe the role of the algebraic properties at a greater detail.

Let us now try to prove some properties of the features and their interactions. We begin by stating a few results for regular sets, (which in our case apply to features) from Kuratowski[KM76], Requicha and Tilove[RT78]

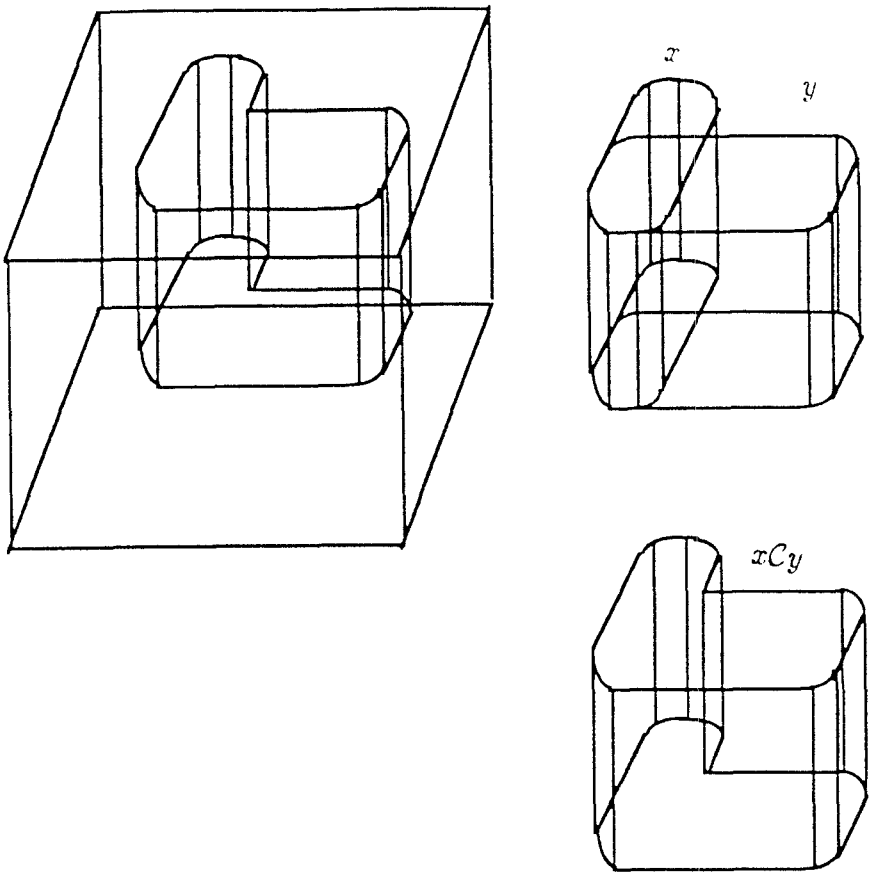


Figure 3.13: An example ilustrating the combination operation.

Theorem 1 *(Requicha and Tilove). The regular sets are a* boolean algebra *with operations* $\cup^*$, $\cap^*$, *and* c^*; *i.e., they satisfy the properties stated below. Let* X, Y *and* Z *be arbitrary regular sets,* W *the universal set and* $\emptyset$ *the empty set.*

1. *Union and intersection are commutative:*

$$X \cup^* Y = Y \cup^* X;$$

$$X \cap^* Y = Y \cap^* X.$$

2. *Each of the operations union and intersection are distributive over the other.*

$$X \cup^* (Y \cap^* Z) = (X \cup^* Y) \cap^* (X \cup^* Z);$$

$$X \cap^* (Y \cup^* Z) = (X \cap^* Y) \cup^* (X \cap^* Z).$$

3. *The empty set* $\emptyset$ *and the universal set* W *are identity elements for the union and intersection operations:*

$$X \cup^* \emptyset = X;$$

$$X \cap^* W = X.$$

4. *The complement satisfies the following properties:*

$$X \cup^* c^* X = W;$$

$$X \cap^* c^* X = \emptyset.$$

Proposition 2 *(Requicha and Tilove). If* X *and* Y *are regular sets, then*

$$X \cup^* Y = X \cup Y.$$

Proposition 3 *(Requicha and Tilove). If* X *and* Y *are regular,*

$$X -^* Y = X \cap^* cY = X \cap^* c^* Y.$$

The next two results are from Kuratowski [KM76] (with slight modification).

Proposition 4 *(Kuratowski). The regularized union operation on regular sets is associative.*

Proposition 5 *(Kuratowski). The regularized intersection operation on regular sets is associative.*

The following proposition is useful in showing the *volume-preserving* property of maximal extension (see Proposition 13). Additional motivation for this proposition is given later in this section.

Proposition 6 *Given a feature* x *and a patch* p *of* x,

$$ps(x) \subseteq \mathcal{I}_p(x).$$

Proof:

$$\mathcal{I}_p(x) = (\mathcal{B}_p(x) \cap \mathcal{C}_p(x)) \cup \mathrm{ps}(x)$$

Therefore,

$$\mathrm{ps}(x) \subseteq \mathcal{I}_p(x)$$

□

The following proposition shows the associativity property of the truncation operation. Therefore, if a feature were to be computed as $(x\mathcal{T}y)\mathcal{T}z$, then we need not recompute it as $(x\mathcal{T}z)\mathcal{T}y$.

Proposition 7 *For any three features x, y and z, the following result holds:*

$$(ps(x) -^* ps(y)) -^* ps(z) = (ps(x) -^* ps(z)) -^* ps(y).$$

Proof:

$$\begin{aligned}
&(\mathrm{ps}(x) -^* \mathrm{ps}(y)) -^* \mathrm{ps}(z) \\
&\quad = (\mathrm{ps}(x) \cap^* c^*\mathrm{ps}(y)) \cap^* c^*\mathrm{ps}(z) \quad \text{(from Proposition 3)} \\
&\quad = \mathrm{ps}(x) \cap^* c^*\mathrm{ps}(y) \cap^* c^*\mathrm{ps}(z) \quad \text{(from Proposition 5)}
\end{aligned}$$

Similarly,

$$(\mathrm{ps}(x) -^* \mathrm{ps}(z)) -^* \mathrm{ps}(y) = \mathrm{ps}(x) \cap^* c^*\mathrm{ps}(y) \cap^* c^*\mathrm{ps}(z).$$

□

Proposition 8 *Given a feature x and a patch p of x, if $\mathcal{I}_p(x) = ps(x)$, then $ps(x\mathcal{M}_p y) = ps(x)$, for any feature y.*

Proof: Since $\mathcal{I}_p(x) = \mathrm{ps}(x)$, $\mathcal{I}_p(x) \neq$ INVALID. Therefore,

$$x\mathcal{M}_p y = \langle \mathrm{ps}(x\mathcal{M}_p y), v \rangle,$$

where $\mathrm{ps}(x\mathcal{M}_p y) = \mathcal{I}_p(x) \cap^* (\mathrm{ps}(x) \cup^* \mathrm{ps}(y))$.

$$\begin{aligned}
\mathrm{ps}(x\mathcal{M}_p y) &= \mathcal{I}_p(x) \cap^* (\mathrm{ps}(x) \cup^* \mathrm{ps}(y)) \\
&= \mathrm{ps}(x) \cap^* (\mathrm{ps}(x) \cup^* \mathrm{ps}(y)) \\
&= \mathrm{ps}(x) \cap^* (\mathrm{ps}(x) \cup \mathrm{ps}(y)) \quad \text{(from Proposition 2)} \\
&= ki(\mathrm{ps}(x) \cap (\mathrm{ps}(x) \cup \mathrm{ps}(y))) \\
&= ki(\mathrm{ps}(x)) \\
&= \mathrm{ps}(x)
\end{aligned}$$

□

Since the infinite extension, for most features and patches is an infinite solid, one might doubt the applicability of the above proposition. However, one might note that, in any implementation we delimit the infinite extension at the boundaries of the stock; and therefore infinite extension will be a finite solid. Thus, there will be a lot of cases where $\mathcal{I}_p(x) = \mathrm{ps}(x)$. In all such cases, we never need to compute $x\mathcal{M}_p y$ for any feature y or any patch p, because $\mathrm{ps}(x\mathcal{M}_p y) = \mathrm{ps}(x)$. This proposition is illustrated through an example in Section 3.7.

The following proposition is used in proving Proposition 10. Proposition 10 provides an easy way to detect the cases where $\mathrm{ps}(x\mathcal{T}y) = \mathrm{ps}(x)$ in the domain of convex features. In such cases, we need not compute $x\mathcal{T}y$, since it will not result in a new feature. The proposition states that if we can find a patch p of x that is **ANTI** with respect to y, then we can infer $\mathrm{ps}(x) -^* \mathrm{ps}(y) = \mathrm{ps}(x)$ and $\mathrm{ps}(y) -^* \mathrm{ps}(x) = \mathrm{ps}(y)$.

Proposition 9 *Given two features x and y, if $ps(x)$ and $ps(y)$ are convex sets, and if a patch p of x is* **ANTI** *$ps(y)$, then*

$$ps(x) \cap^* ps(y) = \emptyset$$

Proof: At every point p_i on the patch p consider the tangent planes to $\mathrm{ps}(x)$ and $\mathrm{ps}(y)$, and the half-spaces G_i and H_i that contain $\mathrm{ps}(x)$ and $\mathrm{ps}(y)$ respectively. Since the patch p is **ANTI** $\mathrm{ps}(y)$, at any point $p_i \in p$ $G_i \cap^* H_i = \emptyset$. Therefore,

$$\bigcap_{p_i} G_i \cap^* \bigcap_{p_i} H_i = \emptyset$$

for all $p_i \in p$. Since each $G_i \supseteq \mathrm{ps}(x)$ and $H_i \supseteq \mathrm{ps}(y)$, $\mathrm{ps}(x) \cap^* \mathrm{ps}(y) = \emptyset$.

□

Proposition 10 *Given two features x and y, if $ps(x)$ and $ps(y)$ are convex sets, and if a patch p of x is* **ANTI** *$ps(y)$, then*

$$ps(x) -^* ps(y) = ps(x) \text{ and } ps(y) -^* ps(x) = ps(y)$$

Proof: The result follows directly from Proposition 9.

□

Proposition 11 *Given features x, y, z, u and v and patches a, b and c such that $z\mathcal{M}_a y = u$, and $y\mathcal{M}_b x = v$, if $\mathcal{I}_a(z) = \mathcal{I}_c(u)$ and $(\mathcal{I}_a(z) \cap^* \mathcal{I}_b(y)) \supseteq (\mathcal{I}_a(z) \cap^* ps(x))$, then, the following result holds:*

$$ps(z\mathcal{M}_a v) = ps(u\mathcal{M}_c x).$$

Proof:

$$\begin{aligned}
&\mathrm{ps}(z\mathcal{M}_a v) \\
&= \mathcal{I}_a(z) \cap^* (\mathrm{ps}(z) \cup^* \mathrm{ps}(v)) \\
&= \mathcal{I}_a(z) \cap^* (\mathrm{ps}(z) \cup^* (\mathcal{I}_b(y) \cap^* (\mathrm{ps}(y) \cup^* \mathrm{ps}(x)))) \\
&= \mathcal{I}_a(z) \cap^* (\mathrm{ps}(z) \cup^* ((\mathcal{I}_b(y) \cap^* \mathrm{ps}(y)) \cup^* (\mathcal{I}_b(y) \cap^* \mathrm{ps}(x)))) \\
&\quad \text{(from Theorem 1)} \\
&= \mathcal{I}_a(z) \cap^* (\mathrm{ps}(z) \cup^* (\mathcal{I}_b(y) \cap^* \mathrm{ps}(y)) \cup^* (\mathcal{I}_b(y) \cap^* \mathrm{ps}(x))) \\
&\quad \text{(from Proposition 4)} \\
&= \mathcal{I}_a(z) \cap^* (\mathrm{ps}(z) \cup^* \mathrm{ps}(y) \cup^* (\mathcal{I}_b(y) \cap^* \mathrm{ps}(x))) \\
&\quad \text{(from Proposition 6)} \\
&= (\mathcal{I}_a(z) \cap^* \mathrm{ps}(z)) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(y)) \cup^* (\mathcal{I}_a(z) \cap^* \mathcal{I}_b(y) \cap^* \mathrm{ps}(x)) \\
&\quad \text{(from Theorem 1)} \\
&= \mathrm{ps}(z) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(y)) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(x)) \\
&\quad \text{(from Proposition 6)}
\end{aligned}$$

Therefore,

$$\mathrm{ps}(z\mathcal{M}_a v) = \mathrm{ps}(z) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(y)) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(x)) \tag{3.2}$$

$$\begin{aligned}
&\mathrm{ps}(u\mathcal{M}_c x) \\
&= \mathcal{I}_c(u) \cap^* (\mathrm{ps}(u) \cup^* \mathrm{ps}(x))
\end{aligned}$$

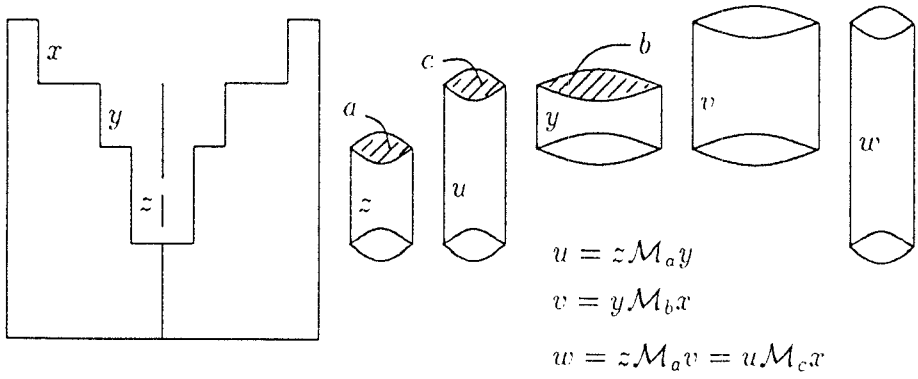


Figure 3.14: Example illustrating Proposition 11.

$$
\begin{aligned}
&= \mathcal{I}_c(u) \cap^* ((\mathcal{I}_a(z) \cap^* (\mathrm{ps}(z) \cup^* \mathrm{ps}(y))) \cup^* \mathrm{ps}(x)) \\
&= \mathcal{I}_c(u) \cap^* ((\mathcal{I}_a(z) \cap^* \mathrm{ps}(z)) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(y)) \cup^* \mathrm{ps}(x)) \\
&\quad \text{(from Theorem 1)} \\
&= \mathcal{I}_a(z) \cap^* (\mathrm{ps}(z) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(y)) \cup^* \mathrm{ps}(x)) \\
&\quad \text{(from Proposition 6)} \\
&= (\mathcal{I}_a(z) \cap^* \mathrm{ps}(z)) \cup^* (\mathcal{I}_a(z) \cap^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(y))) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(x)) \\
&\quad \text{(from Theorem 1)} \\
&= \mathrm{ps}(z) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(y)) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(x)) \\
&\quad \text{(from Proposition 6)}
\end{aligned}
$$

$$\mathrm{ps}(u\mathcal{M}_c x) = \mathrm{ps}(z) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(y)) \cup^* (\mathcal{I}_a(z) \cap^* \mathrm{ps}(x)) \tag{3.3}$$

The result follows from equations 3.2 and 3.3.

□

We will illustrate the above proposition through an example. Consider the features x, y and z as shown in Figure 3.14, where x is a rectangular pocket and y and z are holes. The above proposition shows that $\mathrm{ps}(z\mathcal{M}_a v) = \mathrm{ps}(u\mathcal{M}_c x)$. Therefore, if one of them has been computed, we need not compute the other. This can also be thought of as showing the associativity property of maximal extension in certain cases.

Let us consider a piece of stock, S_0 and let S_n be the final part obtained by successively subtracting n features, f_1, f_2, ..., f_n from S_0. Thus,

$$S_n = S_0 -^* \bigcup_{i=1}^{n} \mathrm{ps}(f_i)$$

Using the feature algebra we can obtain new features by applying the operations pairs of features. Let us consider a pair of features $\langle x, y \rangle$ and an operation η. Typically, the way we use the algebra is to replace a feature x in a feature set (a set of features describing a part, also called a feature interpretation) by $x\eta y$ provided $x\eta y$ is not INVALID. This replacement must preserve the shape of the part i.e. the space occupied by all the features combined should remain unchanged. One way, we can guarantee this is if we show that $\mathrm{ps}(x) \cup \mathrm{ps}(y) = \mathrm{ps}(x\eta y) \cup \mathrm{ps}(y)$ for all the operations η that we have defined. We will call this property as the *volume preserving* property of an operation. The following propositions show that this is the case for the $\mathcal{T}$, $\mathcal{M}_p$ and $\mathcal{C}$. However, this property

does not hold for $\mathcal{M}_p^2$, (using the example given in Figure 3.12 one can easily construct an example to show this) and hence this operation must be used with circumspection.

Proposition 12 *Given two features x and y, if $x\mathcal{T}y \neq INVALID$, then, the following result holds:*

$$ps(x) \cup ps(y) = ps(x\mathcal{T}y) \cup ps(y)$$

Proof:

$$\begin{aligned}
&\mathrm{ps}(x\mathcal{T}y) \cup \mathrm{ps}(y) \\
&\quad = (\mathrm{ps}(x) -^* \mathrm{ps}(y)) \cup \mathrm{ps}(y) \\
&\quad = (\mathrm{ps}(x) \cap^* c^*\mathrm{ps}(y)) \cup \mathrm{ps}(y) \quad \text{(from Proposition 3)} \\
&\quad = (\mathrm{ps}(x) \cup^* \mathrm{ps}(y)) \cap^* (c^*\mathrm{ps}(y) \cup^* \mathrm{ps}(y)) \\
&\qquad \text{(from Theorem 1 and Proposition 2)} \\
&\quad = \mathrm{ps}(x) \cup^* \mathrm{ps}(y) \quad \text{(from Theorem 1)} \\
&\quad = \mathrm{ps}(x) \cup \mathrm{ps}(y) \quad \text{(from Proposition 2)}
\end{aligned}$$

□

Proposition 13 *Given two features x and y, if $x\mathcal{M}_p y \neq INVALID$, then, the following result holds:*

$$ps(x) \cup ps(y) = ps(x\mathcal{M}_p y) \cup ps(y)$$

Proof:

$$\begin{aligned}
&\mathrm{ps}(x\mathcal{M}_p y) \cup \mathrm{ps}(y) \\
&\quad = (\mathcal{I}_p(x) \cap^* (\mathrm{ps}(x) \cup^* \mathrm{ps}(y))) \cup \mathrm{ps}(y) \\
&\quad = ((\mathcal{I}_p(x) \cap^* \mathrm{ps}(x)) \cup^* (\mathcal{I}_p(x) \cap^* \mathrm{ps}(y))) \cup \mathrm{ps}(y) \quad \text{(from Theorem 1)} \\
&\quad = ((\mathcal{I}_p(x) \cap^* \mathrm{ps}(x)) \cup^* (\mathcal{I}_p(x) \cap^* \mathrm{ps}(y))) \cup^* \mathrm{ps}(y) \quad \text{(from Proposition 2)} \\
&\quad = (\mathrm{ps}(x) \cup^* (\mathcal{I}_p(x) \cap^* \mathrm{ps}(y))) \cup^* \mathrm{ps}(y) \quad \text{(from Proposition 6)} \\
&\quad = \mathrm{ps}(x) \cup^* \mathrm{ps}(y) \cup^* (\mathcal{I}_p(x) \cap^* \mathrm{ps}(y)) \quad \text{(from Proposition 4)} \\
&\quad = \mathrm{ps}(x) \cup \mathrm{ps}(y) \cup (\mathcal{I}_p(x) \cap^* \mathrm{ps}(y)) \quad \text{(from Proposition 2)} \\
&\quad \supseteq \mathrm{ps}(x) \cup \mathrm{ps}(y)
\end{aligned}$$

Therefore,

$$\mathrm{ps}(x\mathcal{M}_p y) \cup \mathrm{ps}(y) \supseteq \mathrm{ps}(x) \cup \mathrm{ps}(y) \tag{3.4}$$

$$\begin{aligned}
&\mathrm{ps}(x\mathcal{M}_p y) \cup \mathrm{ps}(y) \\
&\quad = (\mathcal{I}_p(x) \cap^* (\mathrm{ps}(x) \cup^* \mathrm{ps}(y))) \cup \mathrm{ps}(y) \\
&\quad = (\mathcal{I}_p(x) \cap^* (\mathrm{ps}(x) \cup^* \mathrm{ps}(y))) \cup^* \mathrm{ps}(y) \quad \text{(from Proposition 2)} \\
&\quad \subseteq (\mathrm{ps}(x) \cup^* \mathrm{ps}(y)) \cup^* \mathrm{ps}(y) \\
&\quad \subseteq \mathrm{ps}(x) \cup \mathrm{ps}(y) \quad \text{(from Proposition 2)}
\end{aligned}$$

$$\mathrm{ps}(x\mathcal{M}_p y) \cup \mathrm{ps}(y) \subseteq \mathrm{ps}(x) \cup \mathrm{ps}(y) \tag{3.5}$$

The result follows from equations 3.4 and 3.5.

□

Proposition 14 *Given two features x and y, if $x\mathcal{C}y \neq INVALID$, then, the following result holds:*

$$ps(x) \cup ps(y) = ps(x\mathcal{C}y) \cup ps(y)$$

Proof:

$$\begin{aligned}
&\mathrm{ps}(x\mathcal{C}y) \cup \mathrm{ps}(y) \\
&= (\mathrm{ps}(x) \cup^* \mathrm{ps}(y)) \cup \mathrm{ps}(y) \\
&= (\mathrm{ps}(x) \cup \mathrm{ps}(y)) \cup \mathrm{ps}(y) \quad \text{(from Proposition 2)} \\
&= \mathrm{ps}(x) \cup \mathrm{ps}(y)
\end{aligned}$$

□

3.5 Feature Redundancy and Subsumption

Let us consider a set of features, f_i $1 \leq i \leq n$ describing a part. As we have already seen, the final part S_n is given by

$$S_n = S_0 -^* \bigcup_{i=1}^{n} \mathrm{ps}(f_i)$$

where S_0 is the stock.

From the above equation, we can see that if $\mathrm{ps}(f_l) \subseteq \mathrm{ps}(f_m)$ for some $1 \leq l, m \leq n$ and $l \neq m$, then we can well describe the part equally well by the features $\{f_i\}$ $1 \leq i \leq n$ and $i \neq l$. Thus, the feature f_l is redundant and can be eliminated from the feature interpretation. This kind of feature redundancy is given in formal terms by a relation called *subsumption*, denoted by $\mathcal{S}$.

Given two features x and y, then $x\mathcal{S}y$ is true iff $\mathrm{ps}(x) \supseteq \mathrm{ps}(y)$ and false otherwise. If $x\mathcal{S}y$ is true, then we say that x *subsumes* y. It may be noted that subsumption is a transitive relation.

If a feature x subsumes a feature y in a feature interpretation FI, then FI - $\{y\}$, is another valid representation of the final part. Since FI - $\{y\}$ has fewer features than FI, it can be thought of as a simpler representation of the part.

In the existing implementation of the feature algebra, we check for subsumption between pairs of features in a feature interpretation. Features that are subsumed by at least one other feature are deleted from the feature interpretation of the part.

Let us make a few observations regarding other kinds of feature redundancy:

1. Suppose, we have a feature x in a feature interpretation FI such that there is no other feature in FI that subsumes x. However, if we have a pair of features u, v such that $u \neq x$ and $v \neq x$ and $(\mathrm{ps}(u) \cup \mathrm{ps}(v)) \supseteq \mathrm{ps}(x)$, then we can eliminate x from FI and still have a valid feature interpretation of the part. In more general terms, we can think of a collection of features making a different collection of features redundant. When we have such kinds of feature redundancy, (with features belonging to several collections of features) it is hard to determine what features need to be eliminated. Furthermore, such extensive testing for redundancy is computationally very expensive. Such kinds of feature redundancy are not handled by the current implementation of the feature algebra.

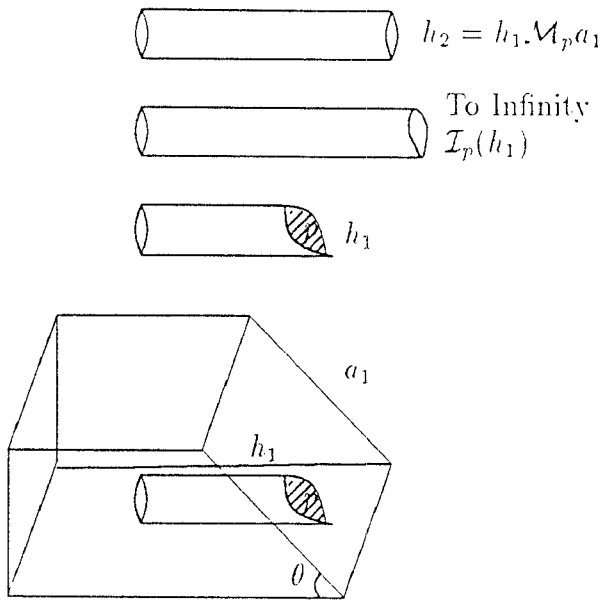


Figure 3.15: A part with an angle and a hole.

2. One can think of feature redundancy in a different way. Suppose all the patches of a feature are labeled UNBLOCKED: then the boundary of the feature separates air and air. Hence, the feature is contained in other features and is redundant in any feature interpretation of the part. This kind of feature redundancy can be described as a special case of the feature redundancy discussed in the previous paragraph. However, we are discussing this separately because this kind of feature redundancy is computationally easy to test. The current implementation of the feature algebra does not handle this kind of feature redundancy either.

3.6 Applications of the Operations

As we have seen from Propositions 12, 13 and 14 if we replace a pair of valid features $\langle x, y \rangle$, by a pair of valid features $\langle x \eta y, y \rangle$ (where η is one of the operations discussed earlier,) the shape of the part is not changed. Let us see how this can be used in addressing feature interactions.

Consider the part shown in Figure 1.5, which is reproduced in Figure 3.15 with additional details. The figure shows a part with two features; a hole h_1 and an angle a_1. The patch p on h_1, the infinite extension of h_1 with respect to the patch p, $\mathcal{I}_p(h_1)$, and the maximal extension into the feature a_1. $h_2 = h_1.\mathcal{M}_p a_1$ are also shown. From this, we see that the maximal extension operation enables us to compute the feature interpretation $\{h_2, a_1\}$, given the interpretation $\{h_1, a_1\}$.

Consider the part shown in Figure 3.16. It is described as a part obtained by subtracting a hole h_1 and a slot s_1, in that order from the initial stock. The feature s_1 divides h_1 into two holes h_2 and h_3. These two holes can be computed using the truncation operation.

3.7 Applications of the Algebraic Properties

Now, we illustrate the use of the algebraic properties through some examples. For the sake of brevity, we will illustrate only the Propositions 7 and 8. Similar examples can be constructed to illustrate the role of the other algebraic properties.

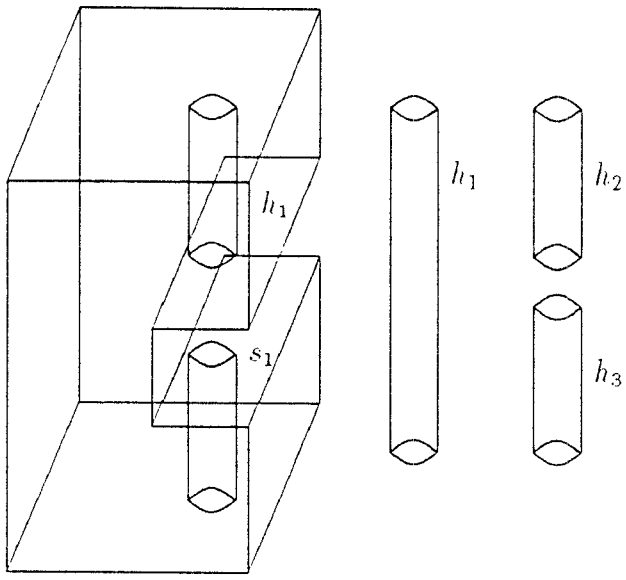


Figure 3.16: An example showing the interaction of a hole with a slot.

Consider the part shown in Figure 1.1. The features in this part are reproduced in Figure 3.17. We compute $h_2 = h_1 \mathcal{T} s_1$ and $h_3 = h_1 \mathcal{T} s_2$ using the operations in the algebra. Hence, $\text{ps}(h_2) = \text{ps}(h_1) -^* \text{ps}(s_1)$ and $\text{ps}(h_3) = \text{ps}(h_1) -^* \text{ps}(s_2)$. Let us now assume that we computed $h_4 = h_2 \mathcal{T} s_2$ using the operations in the algebra. Now, from Proposition 7 we can infer $\text{ps}(h_3 \mathcal{T} s_1) = \text{ps}(h_4)$. Since, we already computed the feature h_4, we need not compute $h_3 \mathcal{T} s_1$.

Now, let us consider the infinite extension of the feature h_1 with respect to the patch a shown in Figure 3.17. $\mathcal{I}_a(h_1)$ is an infinite cylinder. However, since all the features are bounded by the boundaries of the initial stock, we can consider only the portion of infinite extension contained within the initial stock. Under this modified definition, $\mathcal{I}_a(h_1) = ps(h_1)$. Hence, $\text{ps}(h_1 \mathcal{M}_a y) = \text{ps}(h_1)$, for all features y. Therefore, we never need to compute $h_1 \mathcal{M}_a y$ for any feature y.

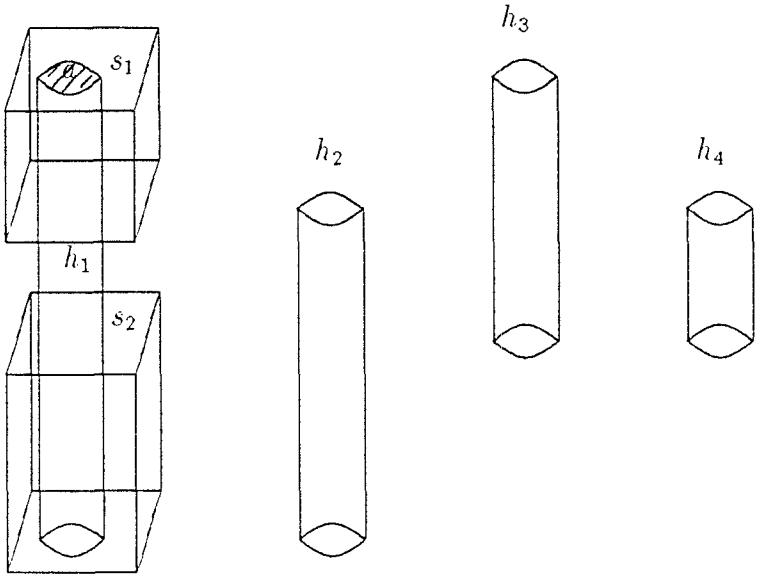


Figure 3.17: The features for the part shown in Figure 1.1 .

Chapter 4

Restricted Feature Algebras

Some restrictions apply.
—Advertisements in the US.

The previous section described an algebra of features; viz., the domain, the operations and the properties. The algebra was defined in a very general way, in an effort to include every feature which could ever be of interest to manufacturing. Currently, we do not have a solid modeler that can handle the entire domain of features that we have defined. Therefore, it is not possible to implement the feature algebra as defined in Chapter 3.

In order to develop algorithms implementing the algebraic operations, we will need to restrict ourselves to some subset of the algebra, by placing restrictions on the features and operators. Ideally, the subset would include all features and interactions of interest in manufacturing—but this seems infeasible since there is no general agreement on what features and interactions these might be. What is considered to be a machinable feature may vary from one manufacturing domain to another, and from one shop floor to another.

Rather than trying to enumerate all features of interest in manufacturing, we instead present a simple example of how a subset of the algebra can be formulated and implemented computationally. For this subset, the domain consists of rectangular solids, cylinders and countersinks that have their planar faces parallel to the faces of the stock. Rectangular solids occur as manufacturing features known by a variety of names, such as a *slot* (which in turn could be *single-ended* or *through*), a *shoulder*, a *pocket*, a *cut-out* and a *notch*.[1] The common manufacturing feature that is cylindrical in shape is a *hole*.

4.1 Computing the Operations

In this section, we will characterize the result of the operations in the algebra. In Chapter 6 the implementation details such as the data structures and the algorithms will be described. In this section, we will describe the result of the operations at a higher level.

[1] In the implementation, we have made the simplifying assumption of pointed corners. However, extending the algebra to include rounded corners is not difficult.

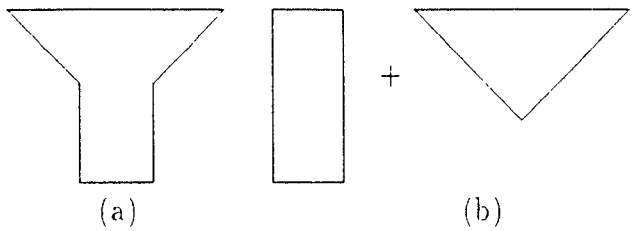


Figure 4.1: Alternative ways of a machining a countersink

4.1.1 Countersink Holes

Figure 4.1(a) shows an example of a countersink hole. A countersink hole can be machined as a single volume as in Figure 4.1(a) or as a hole followed by a taper as in Figure 4.1(b). Figure 3.12 showed how this interaction is captured by the second maximal extension operation. Since the tools required in each case are different we should be able to generate the two volumes in Figure 4.1(b) given the designer specification of a countersink hole. The feature algebra provides both the alternatives, for further evaluation during process planning. However, it is assumed that a countersink feature does not interact with other features in the part.

4.1.2 Truncation

In this section, we will see how the truncation operation can be computed. For the set of features in our domain, $\mathrm{ps}(x) -^* \mathrm{ps}(y)$ could be the point set of a single feature or a pair of features, or it may be a meaningless object as far as this domain is concerned. Figure 4.2 shows examples of the cases that are not of interest; because, $\mathrm{ps}(x) -^* \mathrm{ps}(y)$ is neither a rectangular solid nor a cylinder.

Propagation of labels is straightforward once $\mathrm{ps}(x) -^* \mathrm{ps}(y)$ is computed. This is described in Section 6.8. One way to compute $x\mathcal{T}y$ would be to compute $\mathrm{ps}(x) -^* \mathrm{ps}(y)$ using a solid modeler and then test if it can be considered as the point set of a feature or a pair of features. (We have thus extended $x\mathcal{T}y$ to return either one feature or a pair of features). However, this would be quite inefficient (actual performance statistics are discussed in Section 7.1.6) and would not take advantage of the restricted set of features and interactions that we have here. More efficient methods are described in this section.

If x is a rectangular solid, let $\{x_i | i = 1, \ldots, 6\}$ be the faces of x and if x is a cylinder, let $\{x_i | i = 1, \ldots, 3\}$ be the faces of x (where two of them are planar faces and one is cylindrical face). We will also number the faces such that $x_1 \| x_2$. If x_i is a face, then let x_i' denote the maximal sub-patch of x_i for which $x_i' \mathbf{OUT} y$ is true.

For the cases we have below, the equation for the labels (Equation 3.1) of the patches of $x\mathcal{T}y = \langle u, v \rangle$ can be simplified as follows:

For every patch p in v, if $p \in \mathrm{ps}(x)\mathbf{OUT}\mathrm{ps}(y)$, let p_1 be the patch that contains p

$$\mathrm{label}(p) = \begin{cases} \mathrm{label}(p_1) & \text{if } p \in \mathrm{ps}(x)\mathbf{OUT}\mathrm{ps}(y) \\ \mathrm{UNBLOCKED} & \text{otherwise} \end{cases} \tag{4.1}$$

Now, we describe the procedures for computing $x\mathcal{T}y$.

Case 1: x is a rectangular solid.

Case a. $\mathrm{ps}(x\mathcal{T}y)$ *is two rectangular solids:* (see Figure 4.3) If $x_1 \mathbf{OUT} y$ and $x_2 \mathbf{OUT} y$ and $\forall i \in \{3, \ldots, 6\} x_i'$ consists of two disjoint rectangles, then $\mathrm{ps}(x\mathcal{T}y)$ is two rectangular solids.

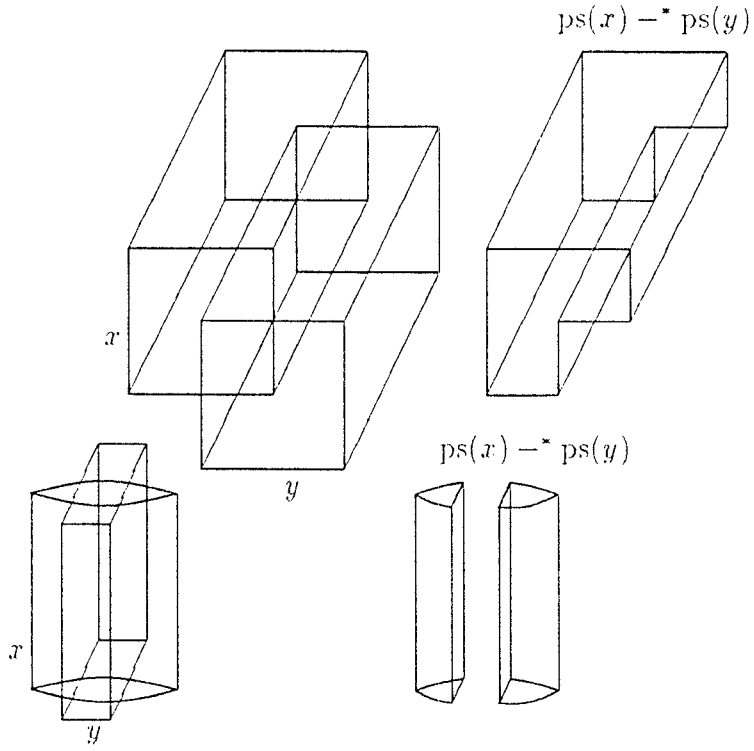


Figure 4.2: Cases where ps(x) $-^*$ ps(y) is not a feature.

Let the two rectangles of x'_i be x_i^1 and x_i^2 where x_i^1 is adjacent to x_1 and x_i^2 is adjacent to x_2. The faces x_1, x_3^1, x_4^1, x_5^1 and x_6^1 determine one rectangular solid. The faces x_2, x_3^2, x_4^2, x_5^2 and x_6^2 determine another rectangular solid. Thus, the two rectangular solids in ps($x\mathcal{T}y$) are determined.

Case b. ps($x\mathcal{T}y$) *is one rectangular solid:* (see Figure 4.4) If $x'_1 = \emptyset$ and $x_2\mathbf{OUT}y$ and $\forall i \in \{3 \ldots 6\} x'_i$ consists of a single rectangle, then ps($x\mathcal{T}y$) is one rectangular solid. The rectangular solid is determined by the faces x_2, x'_3, x'_4, x'_5 and x'_6.

Case 2: x is a cylinder.

Case a. ps($x\mathcal{T}y$) *is two disjoint cylinders:* (see Figure 4.5) If $x_1\mathbf{OUT}y$ and $x_2\mathbf{OUT}y$ and x'_3 consists of two disjoint cylindrical patches, then ps($x\mathcal{T}y$) is two cylinders. Let the two patches of x'_3 be x_3^1 and x_3^2 where x_3^1 is adjacent to x_1 and x_3^2 is adjacent to x_2. The faces x_1 and x_3^1 determine one cylinder. The faces x_2 and x_3^2 determine the another cylinder. Thus, the two cylinders in ps($x\mathcal{T}y$) are determined.

Case b. ps($x\mathcal{T}y$) *is one cylinder:* (see Figure 4.6) If $x'_1 = \emptyset$ and $x_2\mathbf{OUT}y$ and x'_3 consists of a single cylindrical patch, then ps($x\mathcal{T}y$) is one cylinder. The cylinder is determined by the faces x_2 and x'_3.

Case 3: If the conditions for the previous cases are not met, then $x\mathcal{T}y$ =INVALID.

4.1.3 Infinite Extension

In computing infinite extension, it should be stated that in any implementation the original definition can be modified to delimit infinite extension to the boundaries of the stock. Since all the

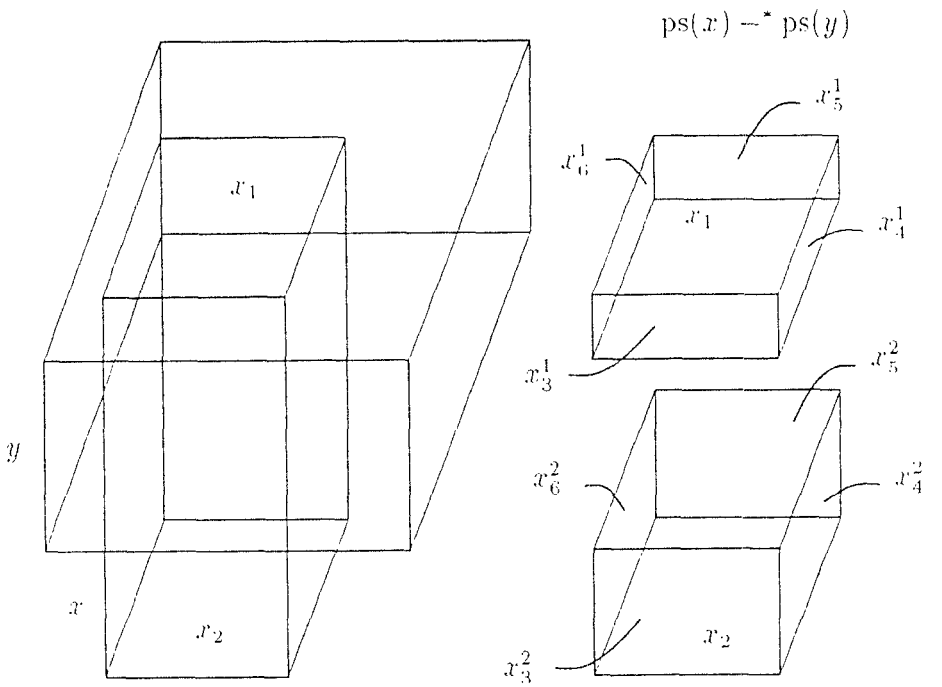


Figure 4.3: Case where $\mathrm{ps}(x) -^* \mathrm{ps}(y)$ is two rectangular solids.

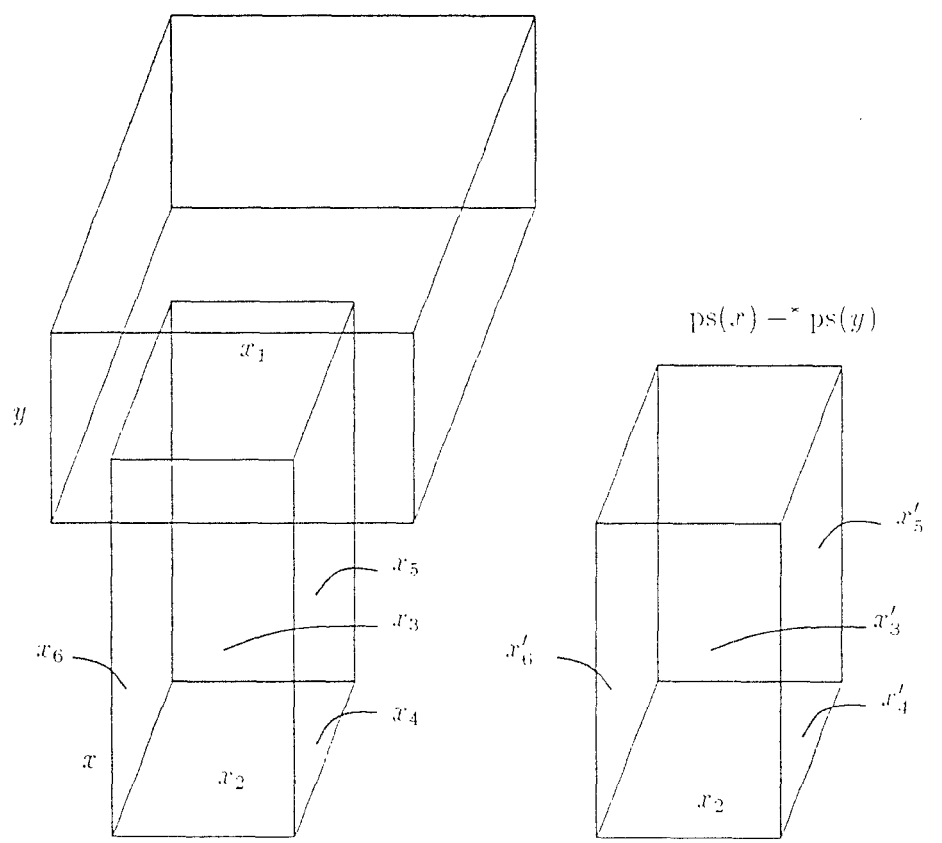


Figure 4.4: Case where $\mathrm{ps}(x) -^* \mathrm{ps}(y)$ is one rectangular solid.

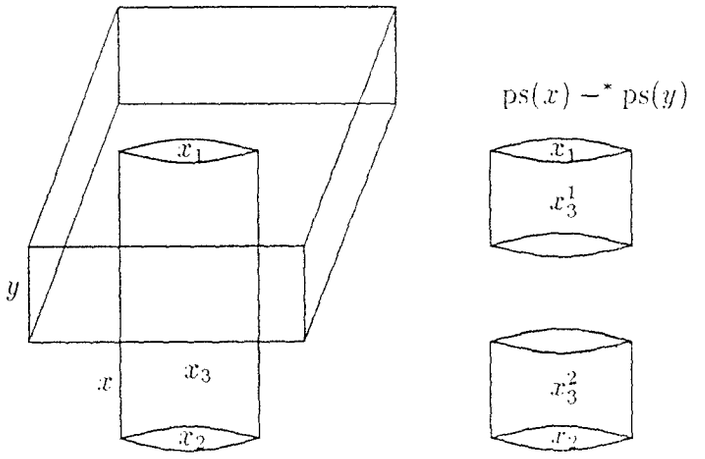


Figure 4.5: Case where $\mathrm{ps}(x) -^* \mathrm{ps}(y)$ is two cylinders.

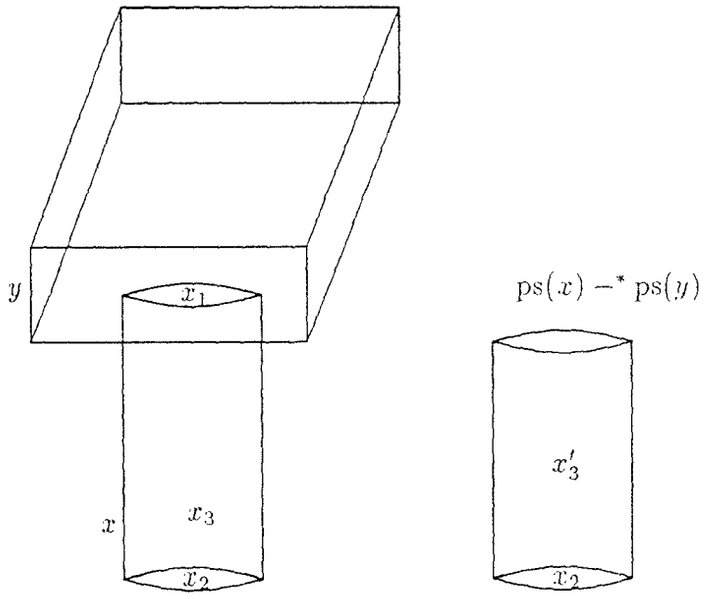


Figure 4.6: Case where $\mathrm{ps}(x) -^* \mathrm{ps}(y)$ is a single cylinder.

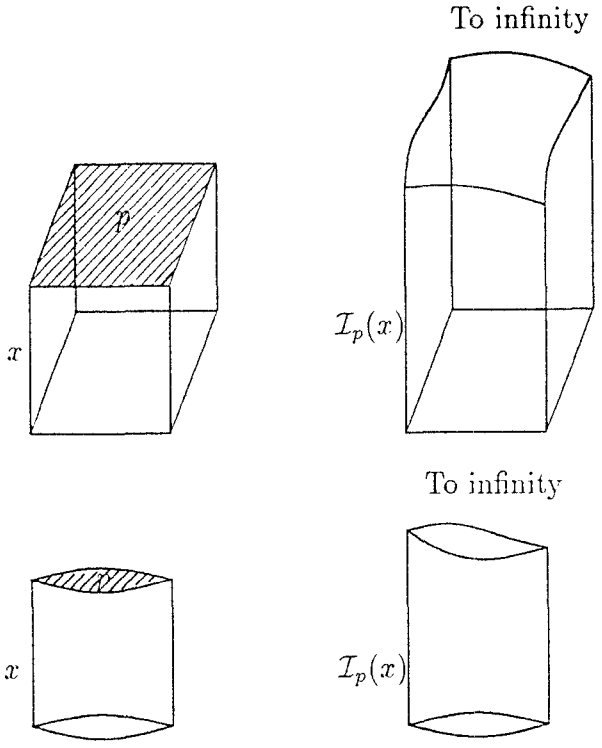


Figure 4.7: Interesting shapes for infinite extension for rectangular solids and cylinders.

features are cut out starting with the stock this does not compromise the generality or the scope of the definition in any way. Given the domain of features we have identified, an important task is to identify the patches of interest for computing infinite extension. We must have as many patches as to include all the interesting shapes, but not too many as to result in repeated computation of the same shape. We should also avoid patches that results in a shape for infinite extension outside our domain. For the restricted domain of rectangular solids and cylindrical holes this task is easy. Typical examples of shapes of interest for infinite extension are shown in Figure 4.7. These shapes can be generated by considering the planar faces of rectangular solids or cylinders as patches. Several other possibilities for patches can be considered but they do not result in shapes of interest. The reader is encouraged to try out various possibilities and be convinced that the above claim is true. In the case of rectangular solids, a proof can be constructed by enumerating all the possible categories of patches. In Figure 4.8 we illustrate two examples of patches that result in infinite extension that is not of interest. In Figure 4.8 (a) the infinite extension is identical to the feature itself and in Figure 4.8 (b) it is too complex to be encompassed by our domain.

Let us call this restricted form of infinite extension as *face infinite extension* ($\mathcal{I}_f$) and the corresponding maximal extension as *face maximal extension* ($\mathcal{M}_f$). Since there are six planar faces in a rectangular solid and two in a cylinder, there are six possible face infinite extensions for a rectangular solid and two for a cylinder. Let us denote the set of all possible face infinite extensions by Σ_I and the set of all possible face maximal extensions by Σ_M. Let us denoted the set of applicable operations by Σ. Therefore, $\Sigma = \{\mathcal{T}\} \cup \Sigma_M$. In the following discussion, given a feature x and a face x_i of x, we will denote the half-space determined by the equation of x_i and containing x as X_i.

Property 1 *If x is a rectangular solid then the faces of $x\mathcal{T}y$ can be written as $\{xs_i | i = 1, \ldots, 6\}$*

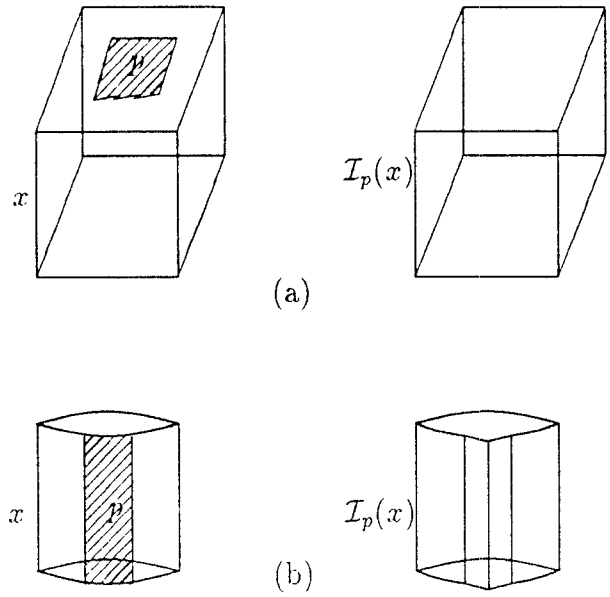


Figure 4.8: Shapes not of interest for infinite extension rectangular solids and cylinders.

where $\forall i \neq 1$ x_i *is in the same plane as* xs_i. *Therefore,* $\mathcal{I}_{xs_1}(x\mathcal{T}y) = X_2 \cap X_3 \cap X_4 \cap X_5 \cap X_6 = \mathcal{I}_{x_1}(x)$. *If* x *is a cylinder, then the faces of* $x\mathcal{T}y$ *can be written as* $\{xs_i | i = 1, \ldots, 3\}$ *where* x_2 *is in the same plane as* xs_2 *and* x_3 *is in the same cylindrical surface as* xs_3. *Therefore,* $\mathcal{I}_{xs_1}(x\mathcal{T}y) = X_2 \cap X_3 = \mathcal{I}_{x_1}(x)$.

4.1.4 Maximal Extension

Property 2 *If the feature* x *is a rectangular solid, and if* $x\mathcal{M}_f y$ *is a valid feature (where* $f = x_1$*), its faces are* f', x_2, x'_3, x'_4, x'_5 *and* x'_6 *where* $f' \| x_2$ *and* x'_i, $\{i = 3 \ldots 6\}$ *is in the same plane as* x_i. *If the feature* x *is a cylinder, and if* $x\mathcal{M}_f y$ *is a valid feature, its faces are* f', x_2, x'_3, *where* $f' \| x_2$ *and* x'_3, *is in the same cylindrical surface as* x_3. *Therefore,* $\mathcal{I}_{f'}(x\mathcal{M}_f y) = \mathcal{I}_f(x)$.

One way to compute $x\mathcal{M}_f y$ given two features x and y is given below:

1. Compute $\mathcal{I}_f(x)$.
2. Using the solid modeler, compute $w = \text{ps}(x) \cup^* \text{ps}(y)$.
3. Using the solid modeler, compute $z = \mathcal{I}_f(x) \cap^* w$.
4. Test if z can be considered as the point set of a feature.

However, this method is very inefficient, and the performance statistics are discussed in Section 7.1.6. A more efficient method is described in the next paragraph.

Let x be a feature whose maximal extension with respect to a planar face $f = x_1$ into a feature y is to be computed. Let f be **IN** or **ANTI** with respect to $\text{ps}(y)$. Otherwise, $x\mathcal{M}_f y$ is INVALID. Let x_2 be the face of x parallel to f. Let y_1 and y_2 be the faces of y parallel to f. Let y_1 be closer to x_2 than y_2. Let $f' = \mathcal{O}(f, y_2)$. The faces f' and x_2 define a solid (a rectangular solid or a cylinder) which determines $\text{ps}(x\mathcal{M}_f y)$. This procedure is illustrated in Figure 4.9. Determining the patches and patch labels after computing $\text{ps}(x\mathcal{M}_f y)$ is described in Section 6.8.

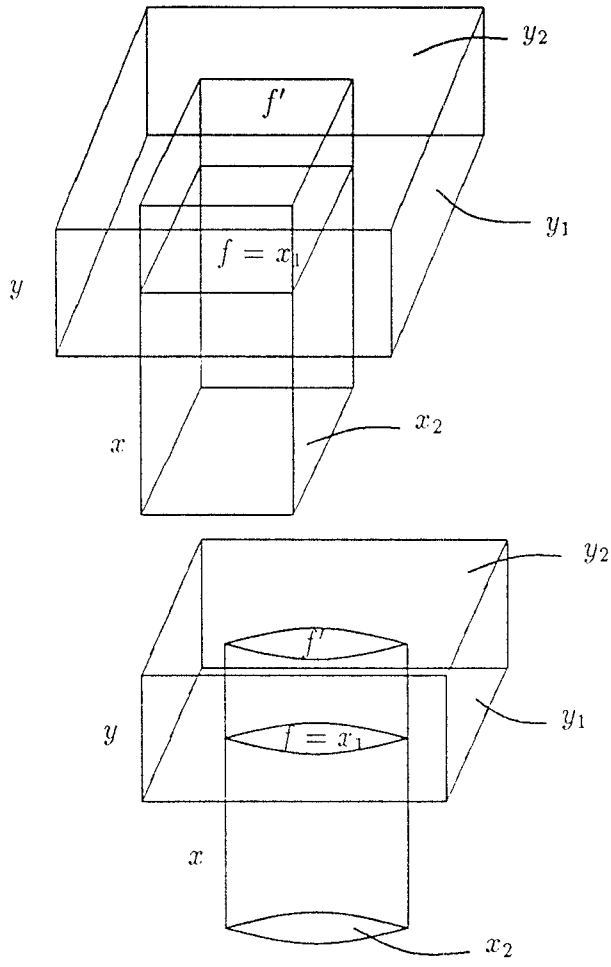


Figure 4.9: Computing maximal extension.

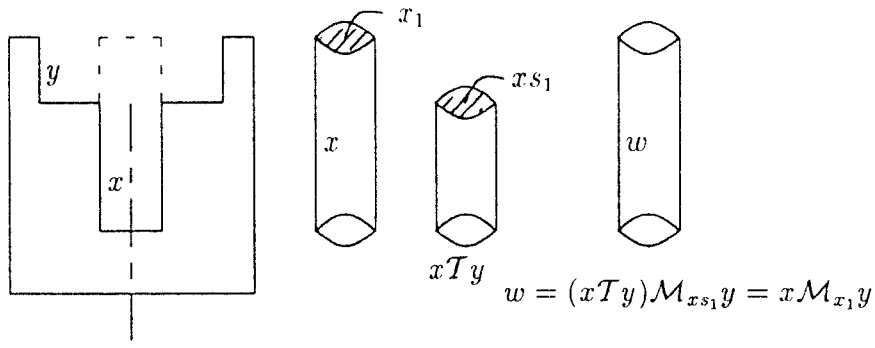


Figure 4.10: Example illustrating the Proposition 15.

4.2 Properties of the Algebra

This section presents some additional properties that hold for the restricted feature algebra that we are dealing with. These properties are restricted to the interactions among cylinders and rectangular solids whose faces are parallel to the faces of the stock. When different kinds of restricted feature algebras are considered different sets of properties hold, and these properties can be used in reducing the computations to be performed in computing new features from the old ones.

Proposition 15 *Given features x, y and xTy, if the face xs_1 of xTy is not in the same plane as any face of x, then*

$$ps((xTy)\mathcal{M}_{xs_1}y) = ps(x\mathcal{M}_{x_1}y)$$

Proof:

$$\begin{aligned}
&\mathrm{ps}((xTy)\mathcal{M}_{xs_1}y) \\
&= \mathcal{I}_{xs_1}(xTy) \cap^* (\mathrm{ps}(xTy) \cup^* \mathrm{ps}(y)) \\
&= \mathcal{I}_{x_1}(x) \cap^* ((\mathrm{ps}(x) -^* \mathrm{ps}(y)) \cup^* \mathrm{ps}(y)) \quad \text{(from Property 1)} \\
&= \mathcal{I}_{x_1}(x) \cap^* ((\mathrm{ps}(x) \cap^* c^*\mathrm{ps}(y)) \cup^* \mathrm{ps}(y)) \quad \text{(from Proposition 3)} \\
&= \mathcal{I}_{x_1}(x) \cap^* ((\mathrm{ps}(x) \cup^* \mathrm{ps}(y)) \cap^* (\mathrm{ps}(y) \cup^* c^*\mathrm{ps}(y))) \quad \text{(from Thm. 1)} \\
&= \mathcal{I}_{x_1}(x) \cap^* (\mathrm{ps}(x) \cup^* \mathrm{ps}(y)) \quad \text{(from Thm. 1)} \\
&= \mathrm{ps}(x\mathcal{M}_{x_1}y)
\end{aligned}$$

□

Figure 4.10 shows an application of the above proposition.

Proposition 16 *Given features x, y, xTy and w, if the face xs_1 of xTy is not in the same plane as any face of x if $\mathcal{I}_{x_1}(x) = ps(x)$ and $ps(w) \cap^* ps(y) = \emptyset$, then*

$$ps((xTy)\mathcal{M}_{xs_1}w) = ps(xTy)$$

Proof:

$$
\begin{aligned}
& \mathrm{ps}(w) \\
&= \mathrm{ps}(w) \cap^* (\mathrm{ps}(y) \cup^* c^* \mathrm{ps}(y)) \ \text{(from Thm. 1)} \\
&= (\mathrm{ps}(w) \cap^* \mathrm{ps}(y)) \cup^* (\mathrm{ps}(w) \cap^* c^* \mathrm{ps}(y)) \ \text{(from Thm. 1)} \\
&= \emptyset \cup^* (\mathrm{ps}(w) \cap^* c^* \mathrm{ps}(y)) \\
&= \mathrm{ps}(w) \cap^* c^* \mathrm{ps}(y)
\end{aligned}
$$

Therefore,

$$\mathrm{ps}(w) = \mathrm{ps}(w) \cap^* c^* \mathrm{ps}(y) \tag{4.2}$$

This result will be used later on in the proof.

$$
\begin{aligned}
& \mathrm{ps}((x \mathcal{T} y) \mathcal{M}_{xs_1} w) \\
&= \mathcal{I}_{xs_1}(x \mathcal{T} y) \cap^* (\mathrm{ps}(x \mathcal{T} y) \cup^* \mathrm{ps}(w)) \\
&= \mathcal{I}_{x_1}(x) \cap^* (\mathrm{ps}(x \mathcal{T} y) \cup^* \mathrm{ps}(w)) \ \text{(from Property 1)} \\
&= \mathrm{ps}(x) \cap^* ((\mathrm{ps}(x) \cap^* c^* \mathrm{ps}(y)) \cup^* \mathrm{ps}(w)) \ \text{(from Prop. 3)} \\
&= \mathrm{ps}(x) \cap^* (\mathrm{ps}(x) \cup^* \mathrm{ps}(w)) \cap^* (\mathrm{ps}(w) \cup^* c^* \mathrm{ps}(y)) \ \text{(from Thm. 1)} \\
&= \mathrm{ps}(x) \cap^* (\mathrm{ps}(x) \cup \mathrm{ps}(w)) \cap^* (\mathrm{ps}(w) \cup^* c^* \mathrm{ps}(y)) \ \text{(from Prop. 2)} \\
&= k(i(\mathrm{ps}(x) \cap (\mathrm{ps}(x) \cup \mathrm{ps}(w)))) \cap^* (\mathrm{ps}(w) \cup^* c^* \mathrm{ps}(y)) \\
&= ki\mathrm{ps}(x) \cap^* (\mathrm{ps}(w) \cup^* c^* \mathrm{ps}(y)) \\
&= \mathrm{ps}(x) \cap^* ((\mathrm{ps}(w) \cap^* c^* \mathrm{ps}(y)) \cup^* c^* \mathrm{ps}(y)) \ \text{(from Eqn. 4.2)} \\
&= \mathrm{ps}(x) \cap^* ((\mathrm{ps}(w) \cup^* c^* \mathrm{ps}(y)) \cap^* c^* \mathrm{ps}(y)) \ \text{(from Thm. 1)} \\
&= \mathrm{ps}(x) \cap^* c^* \mathrm{ps}(y) \\
&= \mathrm{ps}(x) -^* \mathrm{ps}(y) \ \text{(from Prop. 3)} \\
&= \mathrm{ps}(x \mathcal{T} y)
\end{aligned}
$$

□

Figure 4.11 shows an application of the above proposition.

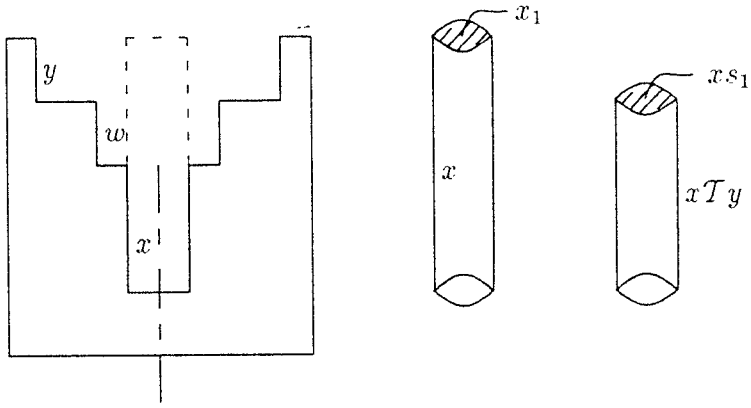


Figure 4.11: Example illustrating Proposition 16.

Chapter 5

The Feature Interface

The practical upshot of all this is that if you stick a Babel fish in your ear you can instantly understand anything said to you in any form or language.
—Douglas Adams, The Hitchhiker's Guide to the Galaxy.

In this chapter, I will describe a user-interface for feature-based design that I built using the Protosolid solid modeler. This interface enables a user to describe a part that needs to be machined. However, it should be noted that the design of the user-interface was not the main focus in implementing the algebra. Even though the current interface is adequate for conveniently specifying the part, a better user-interface will have to be built for use with a more sophisticated feature algebra.

5.1 Protosolid

Protosolid [Van89b, Van89a] is a boundary representation based solid modeler developed at the University of Maryland, College Park. Protosolid is written in Lisp and runs on a Texas Instruments Explorer II, a Lisp machine. Protosolid has a graphical interface for manipulating solids and for visual display. Except the graphical interface, the remainder of Protosolid is written in Common Lisp [Ste84]. Protosolid is a faceted modeler. The faces are embedded in planes and the edges lie on straight lines. Curved analytic surfaces such as cylinders and cones are approximated by a certain number of planar faces. However, in the feature algebra I have implemented all the computations use an exact representation for the cylinders. Thus, the faceted approximation of Protosolid affects only the visual display. Protosolid models planar polyhedra of varying genera, components and connectivity as well as non-manifold solids [RV77, Van89b]. Protosolid also provides a wealth of facilities for querying information about the entities in a solid, iterating over faces, edges, vertices of a solid, etc. These functions were used in developing the feature algebra. Further information about Protosolid and the algorithms used can be found in [Van89b, Van89a].

5.2 A Solid and a Feature

Protosolid uses a variation of the boundary representation data structure (called the *fedge-based* data structure) to represent a solid. I have extended this data structure to deal with features as well. The top-level representation of a feature is as a record structure; more precisely a *structure* in Common Lisp. Table 5.1 shows the fields (also known as slots) in the structure for a feature. Some of the less important fields are omitted in this table. The fields shown in Table 5.1 include all the ones used to describe a solid in Protosolid plus the last five fields, that are relevant only for features.

The first five fields are self-explanatory, except for the phrase *unordered relationship*, which we will discuss briefly. Adjacency relationships between the topological entities in a boundary representation of a solid can either be ordered or unordered. Ordering refers to some kind of (natural) spatial ordering among the related elements. For example, in a non-manifold solid consider the faces incident at an edge. In an unordered relationship we group all these faces as a set. However, in an ordered relationship we order them radially around the edge. The adjacency relationships were originally developed for manifold topologies by Baer, Eastman and Henrion [BEH79]. Various kinds of ordered and unordered relationships are explained in Weiler [Wei86].

The field 'so-name' (the prefix 'so' is an abbreviation for solid) gives the creation history of a feature. For the features given by the user this will simply be a string. However, for features derived using the operations in the algebra the 'so-name' is a prefix expression (in a list format), giving its complete creation history. For example, let A and B be two operations and f_1 and f_2 be two features. Then, (so-name f_1) will be f_1, and (so-name f_2) will be f_2. However, (so-name f_3), where f_3 is $f_1 A f_2$ will be $(A\ f_1\ f_2)$ and (so-name f_4), where f_4 is $f_3 B f_1$ will be $(B\ (A\ f_1\ f_2)\ f_1)$.

The field 'so-extent' gives the smallest (bounding) box enclosing the feature. The patches (and the patch labels) of a feature are given by the field 'so-patches'. This is explained in detail in Section 5.5. The field 'so-params' gives the parameters of a feature. Since Protosolid does not have an exact representation of curved surfaces, this field enables the feature algebra to handle cylinders and countersink features correctly.

All solids in Protosolid are stored on a stack. This means that both features as well solids that are not features (such as intermediate part shapes) are stored on the stack. To facilitate easy access of the features, the feature algebra maintains a hash table of all the features. The field 'so-feat-symbol' is an index into this hash table.

The feature algebra categorizes features into a few types such as rectangular solid, cylinder, frustum of a cone and countersink. The feature type is stored in the field 'so-type'. The 'so-pp-info' field contains the information about a feature that is relevant for process planning. The feature algebra manipulates this field only in a very limited way. The feature algebra uses a process planning system called EFHA [Tho89] to produce the process plan for a feature. Using the information stored in this field the feature algebra produces the input suitable to invoke EFHA to plan for a feature. This process is explained in Section 6.6.

5.3 Creating a Part

In order to create a part using the feature interface, one must first create a stock, denoted by S_0. A stock must be a rectangular solid and its parameters are shown in Table 5.2. The stock is specified by a corner and the dimensions in increasing X, Y and Z coordinate directions. The dimensions are given as a list of three elements. After a stock has been created, the user must specify a set of features, one after the other. If the part after specifying the features $\{f_j,\ 1 \leq j \leq i\}$ is S_i, the part after specifying the feature f_{i+1} is

Field	Comment
so-faces	The faces bordering the feature. Unordered relationship.
so-nfaces	The number of faces bordering the feature.
so-edges	The edges bordering the feature. Unordered relationship.
so-nedges	The number of edges bordering a feature.
so-vertices	Ordered list of vertices bordering the feature.
so-name	The history of the feature as a list.
so-extent	The smallest box containing the feature
so-patches	The patch label information.
so-params	The parameters of a feature.
so-feat-symbol	The index into the hash table of features.
so-type	The category of a feature
so-pp-info	The process planning information for later use.

Table 5.1: The fields in the data structure for a feature.

$$S_{i+1} = S_i -^* \mathrm{ps}(f_{i+1}).$$

The stock as well as the features can be created interactively. However, once a part has been created, one would like store the part creation sequence in a file, so that, in future, the part creation can be *replayed* from the file. Facilities for doing this are provided by Protosolid. Notice that, one may include not only the functions for stock and feature creation but also the functions for rendering a solid, zooming on a solid, etc. in a part creation file.

The features that are currently handled by the feature interface are rectangular solids, cylinders and countersinks. The features are created by primitive instancing [Man88]. The required parameters of the features are shown in Table 5.2. In addition to these parameters, all the features have two other optional parameters that give the information relevant for process planning and information about the patches.

A rectangular solid is specified by a counterclockwise loop of points (as viewed from the outside) and a sweep vector which gives the magnitude and direction in which the loop is to be swept. However, for a rectangular solid, this specification is more cumbersome than some simpler schemes such as specifying two diagonally opposite points. However, this scheme was adopted because it can easily be extended to specify more complex features such as contoured pockets as well.

The radius, height and major axis of a cylinder are given by the 'radius', 'height' and 'major-axis' fields respectively. The 'major-axis' field can be one of the three unit vectors $\vec{i}$, $\vec{j}$ or $\vec{k}$. The 'center' field of a cylinder gives a reference point on its major axis. The parameters are interpreted as follows: A cylinder with the parameters 'height' h, 'radius' r, 'center' c and 'major-axis' m is obtained by considering the circle with center c and radius r and sweeping it along the vector $h * m$.

The parameters of a countersink feature are illustrated through Figure 5.1.

5.4 Validity Checks

The feature interface performs certain checks, to make sure that the parameters specified by the user create a valid feature, and that it is validly positioned on the current part. The current part

Entity	Parameter	Comment
Stock	Corner	A corner of the stock
	Dimensions	Dimensions in increasing X, Y and Z
Rectangular Solid	Loop	A counterclockwise loop of points
	Sweep	Sweep vector towards the solid
Cylinder	Height	Height of the cylinder
	Radius	Radius of the cylinder
	Major-axis	Major axis of the cylinder
	Center	The base point on the major axis
Countersink	Height	Total height of the countersink
	Radius	The maximum radius of the countersink
	F-Center	The center point of cone base
	C-Height	Height of the cylindrical portion
	C-Radius	Radius of the cylindrical portion
	C-Axis	Major axis of the cylinder
	C-Center	Base point on the major axis

Table 5.2: The parameters of the stock and the features.

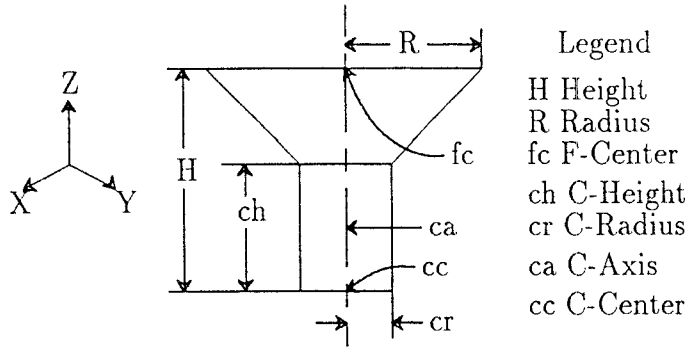


Figure 5.1: The parameters of a countersink hole.

refers to the shape of the part at the time a feature is being created. Some of the validity checks are discussed in this section.

5.4.1 Rectangular Solids

In the case of rectangular solids, the following validity checks are performed:

1. Exactly four points must be specified in the counterclockwise direction in the 'loop' field.
2. The four points must be coplanar and must constitute a valid rectangle.
3. If $l = \langle p_1, p_2, p_3, p_4 \rangle$ is the loop specified by the user (where p_1, p_2, p_3 and p_4 are the points in the loop) and d is the sweep vector, then either the loop l or $l' = \langle p_1 + d, p_2 + d, p_3 + d, p_4 + d \rangle$ must be on a face of the current part.
4. The feature must be of a non-zero volume.
5. The feature must subtract some volume out of the current part.

5.4.2 Cylinders

The following validity checks are performed for cylinders:

1. The center of one of the planar faces of the cylinder must lie on a face of the current part.
2. The parameters must specify a cylinder of non-zero volume.
3. The feature must subtract some volume out of the current part.

5.4.3 Countersinks

The following validity checks are performed for countersinks:

1. The center of the base of the conical portion (see F-Center in Figure 5.1) must lie on a face of the current part.
2. The radius of the conical portion of the countersink must be larger than that of the cylindrical portion.
3. Both the conical and cylindrical portions of the countersink must be of non-zero volume.
4. The feature must subtract some volume out of the current part.

5.5 Representing Patches

The 'so-patches' field of a feature has the information about the geometry and the label of the patches. At the top level, it is organized as list of *face-patches*. Every face of the feature has a face-patch. Therefore, there are six face-patches for a rectangular solid, and three for a cylinder. The data structures for representing the patches of countersink holes are not available in the current version of the feature algebra. A face-patch is a record structure with the fields shown in Table 5.3. The 'label' field of a face-patch indicates whether a face is planar or cylindrical. This should not be confused with the BLOCKED and UNBLOCKED labeling described in Chapter 3. These labels

Field	Comment
Label	The type of the face-patch
Loops	The loop-patches of the face-patch

Table 5.3: The fields of a face-patch.

Field	Comment
Loop-type	The type of the loop-patch
Points	The geometry of the loop-patch
Label	The label (BLOCKED or UNBLOCKED) of the loop-patch

Table 5.4: The fields of a loop-patch.

are given for each loop in a face and are described in the next paragraph. The 'loops' field of a face-patch gives a list of *loop-patches* that belong to the face-patch.

A loop-patch gives the geometry and labeling information for a loop within a face. A loop-patch is a record structure with the fields shown in Table 5.4. The first field 'loop-type' indicates what kind of a loop it is. Currently, it can only be a rectangular loop, a circular loop, or a cylindrical loop. However, as will be discussed later, one will have to have other kinds of loops to represent more complex patches. The field 'label' of a loop-patch indicates whether a loop-patch is BLOCKED or UNBLOCKED. The field 'points' describes the geometry of the loop-patch. If the loop-patch is a rectangular loop, then the 'points' field gives the counterclockwise loop of four points bounding the patch. If the loop-patch is a circular loop, then the 'points' field is a record structure with the fields shown in Table 5.5. These fields give the center and the radius of the patch. If the loop-patch is a cylindrical loop, then the 'points' field is a record structure with the fields shown in Table 5.6. The major-axis, radius and height of a cylindrical patch are given by the 'major-axis', 'radius' and 'height' fields respectively. The 'firstc' and the 'secondc' fields give the centers of the two *closure* faces [DF89] of the cylindrical patch. The fields of a cylindrical patch are illustrated through Figure 5.2.

Let us illustrate the representation of the patches through an example. Consider the part shown in Figure 5.3 that has two features, viz., a slot s and a hole h at the bottom of s. For the sake of brevity, we will show the patches only for the top and the bottom faces of the slot s, which are marked f_1 and f_2 in Figure 5.3. The face f_1 has only one loop; a rectangular loop and the face f_2 has two loops; a rectangular loop and a circular loop with center p_9 and radius r. The face-patches of f_1 and f_2 are shown in a Lisp like syntax, in Figure 5.4 and Figure 5.5 respectively. In the face f_2, the circular loop labeled UNBLOCKED is contained in the rectangular loop BLOCKED. This

Field	Comment
Center	The center of the circular patch
Radius	The radius of the circular patch

Table 5.5: The fields of a circular patch.

Field	Comment
Firstc	An end point on the patch major axis
Secondc	An end point on the patch major axis
Radius	The radius of the cylindrical patch
Height	The height of the cylindrical patch
Major-axis	The major axis of the cylindrical patch

Table 5.6: The fields of a cylindrical patch.

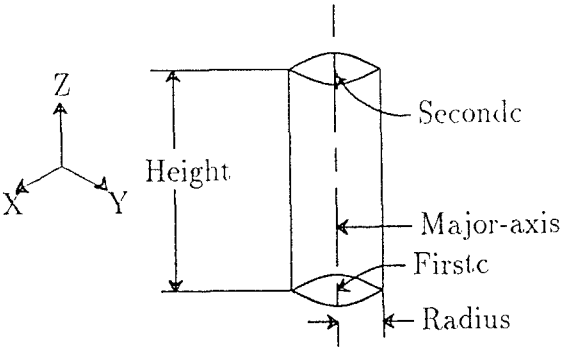


Figure 5.2: The parameters of a cylindrical patch.

type of a situation, wherein one or more UNBLOCKED loops are contained in a BLOCKED loop is very common. However, since we are dealing only with subtractive features, we cannot have a situation where one or more BLOCKED loops is contained in a UNBLOCKED loop.

Even though the above scheme for representing patches can handle nested features, it has some limitations in dealing with all the kinds of patches that occur even in our restricted domain. Consider the part shown Figure 5.6. It has a hole h and a slot s. If we look at the cylindrical face of the hole, part of it is BLOCKED and part of it is UNBLOCKED and our current data structures for a cylindrical loop-patch require a complete cylindrical surface and not a sector. Thus, the patches information for the part in Figure 5.6 cannot be represented.

In Section 8.4.1, we will provide an example of a case where the patches can be specified for a set of features that describe the part, but the patches in all the possible feature interpretations of the part cannot be expressed using the representation we have chosen. Handling patch representation and propagation adequately for a reasonably complex feature algebra is a topic that needs further research.

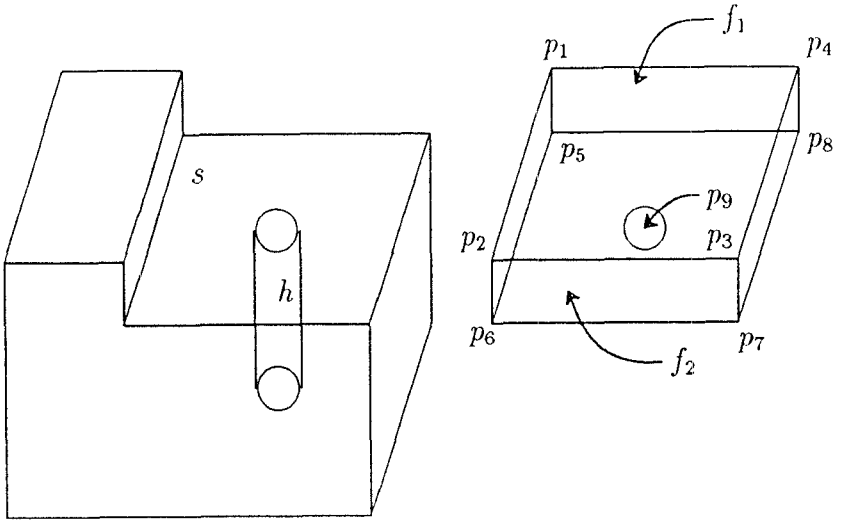


Figure 5.3: A part with a slot and a hole.

```
(make-face-patch      :label   'PLANAR
                      :loops   (LIST
   (make-loop-patch   :loop-type    'RECT-LOOP
                      :points       (LIST p1 p2 p3 p4)
                      :label        'UNBLOCKED )
   ))
```

Figure 5.4: The representation of the patches for the face f_1.

```
(make-face-patch      :label   'PLANAR
                      :loops   (LIST
   (make-loop-patch   :loop-type    'RECT-LOOP
                      :points       (LIST p5 p6 p7 p8)
                      :label        'BLOCKED )
   (make-loop-patch   :loop-type    'RECT-LOOP
                      :points       (make-circular-patch   :center p9
                                                           :radius r )
                      :label        'UNBLOCKED )
   ))
```

Figure 5.5: The representation of the patches for the face f_2.

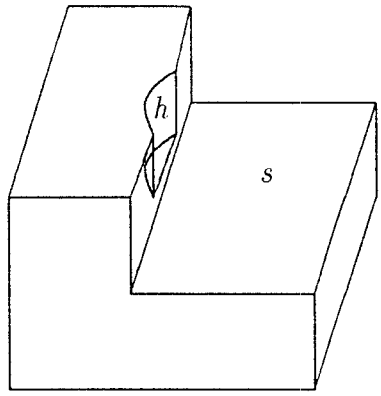


Figure 5.6: An example where the patches of the cylinder h cannot be represented using the current scheme.

Chapter 6

Implementation

Computer implementations quantify task requirements.
—Patrick Henry Winston, Artificial Intelligence.

In this chapter, I will describe the implementation of the feature algebra and its communication with the EFHA [Tho89] process planning system and the Protosolid [Van89b] solid modeler. This implementation has been built as a proof of concept to demonstrate the viability and efficiency of a feature algebra. Implementing a more sophisticated feature algebra that encompasses most of the features relevant to manufacturing is planned for the future.

6.1 Overview of the System

The overview of the system for design and process planning is shown in Figure 6.1. In this figure, all the components except the Protosolid and EFHA were designed and developed by the author. All the components in this block diagram except for the block labeled 'algebraic properties' are written in Lisp [Tex87a, Ste84]. All the code is written in Common Lisp [Ste84], except for the code relating to the user interfaces which is specific to the Explorer series of machines. All the components run on the same machine, viz., a Texas Instruments Explorer II.

The design-by-features interface and the Protosolid solid modeler have already been discussed in Chapter 5. The block labeled 'Feature Algebra' contains all the procedures for performing operations on the features as well as an algorithm to generate alternative feature interpretations of a part described in Section 6.7.

As described in Section 3.4, we have identified certain provable properties of the operations and the features. Using these properties, we can determine the result of an operation on two features without using the procedures to compute the operations. Access to the algebraic properties is optional, i.e., one can use the feature algebra and obtain alternative feature interpretations of a part without accessing the algebraic properties. The algebraic properties are implemented as rules in a dialect of Prolog [CM84, Tex87b] (called the TI Prolog) available on the Texas Instruments Explorer. The module implementing the algebraic properties is discussed in Section 6.5.

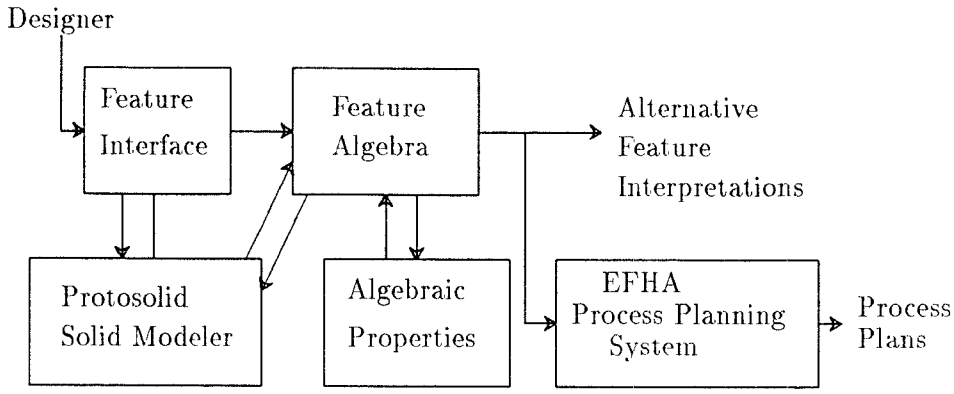


Figure 6.1: The block diagram of the system for design and process planning.

The feature algebra can also optionally invoke the EFHA [Tho89] process planning system in order to produce process plans for the features of the part. The interface to EFHA is described in Section 6.6.

The information flow through the system can be described as follows:

1. The designer describes a part using the design-by-features interface.
2. After all the features have been specified, the designer invokes the algorithm which generates alternative feature interpretations of the part. The procedures for the operations in the algebra are used in this step. The designer has the option to specify whether or not the algebraic properties should be used in this step.
3. Optionally, the algorithm also produces the task specifications suitable for process planning and communicates them to the EFHA process planning system. EFHA comes up with the least cost process plan for a feature (if it exists).

Thus, the final output of the system is all the feature interpretations of a part, and optionally, the process plans for all the features in all the feature interpretations.

6.2 Geometric Primitives

In implementing the restricted feature algebra, two kinds of extensions were made to Protosolid:

1. Extending the data structures used in Protosolid to represent solids. This was discussed in Section 5.2.
2. Adding additional geometric primitives that answer queries relating to solids.

Most of the primitives were added to Protosolid in order to efficiently compute the operations in the algebra. Some of these have been incorporated as general purpose queries accessible to the user. Below, are a few examples of the geometric primitives that were added:

1. Determine if a point is inside, outside or on the boundary of a feature;
2. Determine if a point is inside, outside or on the boundary of a face;
3. Determine the orthogonal projection (see Section 3.2.2) of a point on a face (if it exists);
4. Determine the distance of a point from a line;
5. Determine the point of intersection of two line segments (if it exists).

6.3 Operations

The procedures for computing the operations take advantage of the nature of the features and the interactions in producing the result of an operation. Furthermore, one must realize that only in a small fraction of the cases the application of an operation results in a valid feature(s). Hence, it is important to detect the cases that result in invalid features efficiently. Towards this end, the procedures test for a number of preconditions (or applicability tests) to be satisfied before producing the feature that results from an operation. This contributes to the computational efficiency of the procedures of the algebra. This is discussed with some illustrative examples in Section 7.1.6. Section 6.8 explains how the patches are computed.

Below, we give some examples of preconditions used. Consider the truncation of a hole h_1 by another hole h_2. The following are some of the preconditions that must be satisfied, for $h_1 T h_2$ to be a valid feature and to be distinct from h_1:

1. The major axes of h_1 and h_2 must be the same.
2. The radius of h_2 must be greater than or equal to the radius of h_1.
3. If x is a planar face of h_1 then $x\mathbf{IN}h_2$ or $x\mathbf{WITH}h_2$ or $x\mathbf{OUT}h_2$ must hold.
4. The orthogonal projection of the 'center' parameter of h_1 onto a planar face of h_2 must exist. Let this point be p_1 and the center of the planar face (containing p_1) be p_2.
5. The sum of the distance between p_1 and p_2 and the radius of h_1 must be less than or equal to the radius of h_2.

6.3.1 Finite Precision Arithmetic

While computing the operations, and in numerous instances throughout the restricted feature algebra one is comparing floating point numbers. Since exact arithmetic is not being performed, one must be careful in determining whether or not two floating point numbers are equal. Throughout the implementation of the restricted feature algebra two floating point numbers x and y are considered equal, if $\mid x - y \mid \leq \epsilon$. The ϵ used in Protosolid and the feature algebra is 10^{-12}. Double precision arithmetic (64 bit representation) is used by both Protosolid and the feature algebra in dealing with floating point numbers. Robust ways of computing set operations on solids is a topic of current research [Mil88, Kar88]. This issue has not been addressed by the feature algebra.

6.4 Subsumption

Subsumption can be computed easily using the extents of the features. The subsumption relation between two cylinders can be computed more efficiently by comparing the parameters rather than the extents. Further, for any ordered pair of features $\langle x, y \rangle$, the first time $x \mathcal{S} y$ is computed, it is stored in a boolean array for future reference. This way, we avoid recomputing the subsumption relation more than once for the same pair of features. However, the current implementation of the restricted feature algebra does not take advantage of the transitivity property of subsumption.

6.5 Algebraic Properties

In order to store the relationships between the entities of an algebra, and to infer relationships using the algebraic properties one needs a theorem-prover [Nil80]. This capability was readily available through Prolog on the Explorer machine.

The algebraic properties were implemented as rules in Prolog [CM84, Tex87b]. An example of a rule is shown below:

$$\begin{aligned} trunc(V, Y, W) \ :- \ & atrunc(X, Y, U), X \neq Y, X \neq U, Y \neq U, \\ & atrunc(X, Z, V), X \neq Z, X \neq V, Z \neq V, \\ & atrunc(U, Z, W), U \neq Z, U \neq W, Z \neq W. \end{aligned}$$

The above rule corresponds to Proposition 7.

For the purposes of understanding, the predicates 'trunc' and 'atrunc' can be taken to be the same. Using two different predicates is one way to avoid the problems with *left recursion* [SS86] in Prolog. If the predicate $trunc(X, Y, Z)$ is true, it means that $Z = XTY$.

Initially, as soon as one begins designing a part, all the rules are loaded. Later, during the design of the part, or while computing alternative feature interpretations, one must communicate with the Prolog database, either to assert new facts into the database or to find answers to queries. The interface between Prolog and Lisp on the Explorer provides facilities do all of the above.

6.6 EFHA

The Environment For Hierarchical Abstraction (EFHA) is a frame-based system that can plan for tasks. It can be used in a variety of domains, and one of the domains it is used for is process planning. Using the knowledge about machining processes, EFHA produces plans for individual features. A plan in EFHA is a sequence of steps, where each step is a machining process. EFHA uses the branch and bound search strategy and based on the costs assigned for machining processes it produces the least cost plan to produce a feature. Sometimes, there may not be any feasible plan for making a feature and sometimes there can be several feasible plans. In addition to the cheapest plan, the user can obtain the other plans for making a feature. EFHA is the successor to an earlier system called SIPS (Semi Intelligent Process Selector). Both were developed at the University of Maryland, College Park and the reader is referred to Nau [Nau87] and Thompson [Tho89] for additional details about SIPS and EFHA.

EFHA uses the information about a feature such as its dimensions, tolerances and surface finish in selecting a sequence of processes suitable for machining the feature. Table 6.1 shows an example of the feature parameters needed by EFHA. Table 6.2 shows an example of the process

Feature	Type	Parameter	Value
f_1	Flat Surface	Flatness	0.001
		Positive-Tolerance	0.001
		Negative-Tolerance	0.001
		Surface Finish	50
		Wall Thickness	0.5

Table 6.1: The parameters for a flat surface.

Feature	Process Plan	Cost
f_1	Rough-Face-Mill	1.0
	Rough-Peripheral-Mill	1.5
	Rough-Face-Mill $\rightarrow$ Finish-Face-Mill	3.0

Table 6.2: The process plans for the feature f_1.

plans produced by EFHA for the feature given in Table 6.1. Notice that, several alternative plans are available to machine this feature.

6.7 The Generate-Features Algorithm

Given a set of features that describe a part, one would like to generate alternate sets of features that describe the part. The terms feature interpretation and feature set are used synonymously in this thesis. This section describes an algorithm called generate-features for generating alternate feature sets given one feature set. The algorithm generate-features and the procedures it calls are shown in Figures 6.2 to 6.8. For the sake of simplicity, these procedures assume access to the algebraic properties as well as EFHA. In the actual implementation, both of these are optional. In this algorithm, $\bar{F}$ is the set of feature interpretations generated at any stage, and F is the union of all the features in $\bar{F}$. Initially, F is the starting set of features and $\bar{F} = \{\mathrm{F}\}$. There are two additional variables called OI (old interpretations) and NI (new interpretations) used in this algorithm. In this algorithm, an iteration refers to one execution of the body of the outermost **while** loop. In the beginning of each iteration, the OI is assigned the value of $\bar{F}$ at that instant. During an iteration, more feature interpretations may be found and $\bar{F}$ is updated accordingly. At the end of the iteration, NI is assigned the value of the additional interpretations found in that iteration (which is also the value of $\bar{F} - \mathrm{OI}$). The symbols Σ and Σ_I refer to the set of operations, and the set of infinite extension operations. Σ and Σ_I for the restricted feature algebra are given in Section 4.1.3.

Whenever we compute $x\eta y$, we store the result in an table for quick reference. Therefore, the first step in computing $x\eta y$ is to look in the table, to see if it has a value stored. If there is no value stored, then the algebraic operations and the properties need to be used. The algebraic properties are tried first. If it is possible to determine $x\eta y$ using the properties of the feature algebra, then one need not do any further computations. Otherwise, one has to compute $x\eta y$ using the procedures for the operations.

Once $x\eta y$ is known, it is easy to determine if it is a valid feature or a pair of valid features (see

'valid-feature' and 'valid-feature-pair' in Figure 6.2). The function 'new-ps-feature (x)' returns true if ps-equal (x, y) is false $\forall y \in$ F and false otherwise.

6.7.1 State-Space Formulation

The task of generating all the feature interpretations, given one feature interpretation can be given a state-space [Nil80] formulation . In this formulation, a state is a feature interpretation of the part. A state is a vertex in what is known as the state-space graph. Given two elements x and y in a state s and an operator η, if $x\eta y$ is a valid feature or a pair of valid features, then we can obtain a new feature interpretation s' as follows: if $x\eta y$ is one feature, then $s' = (s - \{x\}) \cup \{x\eta y\}$ and if $x\eta y$ is a pair of features, then $s' = (s - \{x\}) \cup (x\eta y)$. After deriving s' from s, we can draw a directed edge labeled $x\eta y$ from s to s'. By repeated application of the above step we can derive all the possible feature interpretations of the part, linked to each other by directed edges. Such a graph is known as the state-space graph.

It is easy to see that the generate-features algorithm produces the state-space graph. In order to improve the understanding, in Figure 6.9 we show a simplified version of the generate-features algorithm with a graph-search [Nil80] formulation, emphasizing only the derivation of feature interpretations.

6.7.2 Complexity

At first glance, the worst-case complexity of the algorithm might appear to be exponential, because of the possibility of combinatorial explosion if there are several mutually-interacting features. However, geometric locality dictates that each feature will interact with only a few of its neighbors, so it is unlikely that exponential blowup would occur in *real-world* parts. Further, since the interactions occur only among a spatially connected set of features, one can reduce the number of interactions that need to be considered by partitioning the *delta volume* (the union of all the features) into as many disjoint volumes as possible.

Now, we consider an example where a combinatorial explosion of the number of feature interactions occurs. Consider a part that is obtained by removing a slab of material from a piece of stock. A slab is a layer of material of uniform thickness removed from an entire face. Suppose this material has been described by the designer as composed of n slots of uniform thickness. The total number of feature interpretations for this case will be 2^{n-1}. An example of this is shown in Section 7.3 for $n = 4$.

Lemma 1 *The worst-case number of feature interpretations for a part with n features is $\Omega(2^{n-1})$.*

Proof: For the example being considered, let $f_1, f_2, \ldots f_n$ be the n contiguous slots given by the designer. All the possible feature interpretations can be obtained as follows:

One can think of the material of the slab as being separated into n the slots by drawing $n - 1$ hypothetical planes on the slab. One can derive a feature interpretation by choosing k, $0 \leq k < n$ of these $n - 1$ planes. Therefore, the total number of feature interpretations is:

$$\sum_{k=0}^{n-1} \binom{n-1}{k} = 2^{n-1}$$

□

```
procedure generate-features ();
F ← Starting set of features;
F̄ ← {F};
if has-countersinks-p (F) then
  process-countersinks ();
end if ;
for each x in F do
  compute-infinite-extension (x);
end for ;
NI ← F̄;
while NI ≠ ∅ do
  OI ← F̄;
  for each FS in NI do
    for each ⟨x, y⟩ such that x ∈ FS and y ∈ FS and x ≠ y do
      for each η ∈ Σ do
      z ← xηy;
        if valid-feature (z) then
          manipulate-feature (z);
          F̄ ← F̄ ∪ ((FS − {x}) ∪ {z});
        end if ;
        if valid-feature-pair (z) then
          {z1, z2} ← z;
          for each u in {z1, z2} do
            manipulate-feature (u);
          end for ;
          F̄ ← F̄ ∪ ((FS − {x}) ∪ {z1, z2});
        end if ;
        Store the value of xηy in lookup table;
        Store the value of xηy in Prolog database;
      end for ;
    end for ;
  end for :
  NI ← F̄ − OI;
end while ;
Output F, F̄, and EFHA process plans for features in F;
end generate-features;
```

Figure 6.2: The Generate-Features Algorithm.

```
procedure manipulate-feature ( x );
  if new-ps-feature (x) then
    compute-infinite-extension (x);
    F ← F ∪ {x};
    Generate the EFHA task frame for x;
  end if ;
end manipulate-feature;
```

Figure 6.3: Procedure *manipulate-feature*.

```
procedure process-countersinks ();
  local FI;
  FI ← first (F̄); /* The first element of the list */
  for each x in F do
    if countersinkp (x) then
      Breakup countersink into a cone i and a cylinder j;
      F ← F ∪ {i, j};
      FI ← (FI − x) ∪ {i, j};
    end if ;
  end for ;
  F̄ ← F̄ ∪ FI;
end process-countersinks;
```

Figure 6.4: Procedure *process-countersinks*.

```
procedure compute-infinite-extension (x);
  for each I_f in Σ_I do
    compute I_f(x);
    if I_f(x) = ps(x) then
      Assert this fact into Prolog database;
    end if ;
  end for ;
end compute-infinite-extension;
```

Figure 6.5: Procedure *compute-infinite-extension*.

```
procedure has-countersinks-p (x);
  for each y in x do
    if x is a COUNTERSINK feature then
      return (true);
    end if ;
  end for ;
  return (false);
end has-countersinks-p;
```

Figure 6.6: Procedure *has-countersinks-p*.

```
procedure new-ps-feature (x);
  for each y in F do
    if ps-equal (x, y) then return false;
    end if ;
  end for ;
  return true;
end new-ps-feature;
```

Figure 6.7: Procedure *new-ps-feature*.

```
procedure ps-equal (x, y);
  if x and y are of different types then
    return false;
  end if ;
  if the parameters of x and y are equal
    then return true;
    else return false;
  end if ;
end ps-equal;
```

Figure 6.8: Procedure *ps-equal*.

```
procedure ss-generate-features ();
F — Starting set of features;
OPEN— {F};
CLOSED← ∅;
while OPEN ≠ ∅ do
  NEWOPEN— ∅;
  for each FS in OPEN do
  CLOSED— CLOSED ∪ {FS};
    for each ⟨x, y⟩ such that x ∈ FS and y ∈ FS and x ≠ y do
      for each η ∈ Σ do
      z ← xηy;
        if valid-feature (z) then
          NEWOPEN—NEWOPEN ∪ ((FS − {x}) ∪ {z});
        end if ;
        if valid-feature-pair (z) then
          {z1, z2} — z;
          NEWOPEN— NEWOPEN ∪ ((FS − {x}) ∪ {z1, z2});
        end if ;
      end for ;
    end for ;
  end for ;
  OPEN— NEWOPEN;
end while :
Output CLOSED;
/*CLOSED gives the same value as F̄ in generate-features */
end ss-generate-features;
```

Figure 6.9: The state-space formulation of the features algorithm.

6.8 Patches and Patch Labels

In this section, we describe the patches and patch label propagation after computing the shape of the solid for the operations truncation and maximal extension. Throughout this section, x and y denote two arbitrary features.

6.8.1 Truncation

Consider the equation for patch labels given in Section 4.1.2, which is reproduced below for convenience.

If $x\mathcal{T}y = \langle u, v\rangle$, then for every patch p in v, if $p \in \text{ps}(x)\mathbf{OUT}\text{ps}(y)$, let p_1 be the patch of x that contains p.

$$\text{label}(p) = \begin{cases} \text{label}(p_1) & \text{if } p \in \text{ps}(x)\mathbf{OUT}\text{ps}(y) \\ \text{UNBLOCKED} & \text{otherwise} \end{cases}$$

Using the above equation, we can develop an algorithm for deriving the patches and patch labels of $x\mathcal{T}y$ as given below. First, we will see the intuition as to how the above equation can help us, and then the algorithm. Suppose we determine the boundary of $x\mathcal{T}y$. The boundary of $x\mathcal{T}y$ can be split into two kinds of patches. The patches that belong to boundary of $\text{ps}(x)$ (in which case they belong to $\text{ps}(x)\mathbf{OUT}\text{ps}(y)$), and the rest. For a patch that belong to $\text{ps}(x)$ label is same as that of the patch containing it. For any of the rest, the label is UNBLOCKED. Below is an algorithm based on this idea.

Step 1 After $\text{ps}(x\mathcal{T}y)$ has been computed, the faces of $x\mathcal{T}y$ are known. Repeat the Steps 2 to 5 for each face of $x\mathcal{T}y$.

Step 2 Let f be the face under consideration. For each face we have a face-patch. The 'label' field of the face-patch can easily by determined depending on whether the face is planar or cylindrical. Each face-patch comprises of several loop-patches given by the 'loops' field.

Step 3 If there does not exist a face of x that is on the same plane or the same infinite cylinder as f, then consider the entire face-patch as a single loop which is either a rectangular loop, or a circular loop, or a cylindrical loop. Mark its 'loop-type' field accordingly. Mark the 'label' of the loop-patch as UNBLOCKED. Skip the remaining steps for f.

Step 4 Let f' be the face of x chosen in Step 3. Repeat Step 5 for each loop-patch of f'.

Step 5 Let l' be the loop-patch under consideration. If the interior of l' is totally contained in f, add l' to the 'loops' field of the face-patch of f. If the interior of l' is partially contained in f, then split the loop-patch l' into two sub-patches such that one of them is totally contained in f and the other is outside f. However, both the sub-patches have the same 'loop-type' and 'label' fields as l'. Add the sub-patch that is totally contained in f to the 'loops' field of the face-patch of f. (If the interior of l' is outside the face f do nothing.)

Now, we will illustrate the above algorithm through an example. Consider the part shown in Figure 1.1. The features h_1 and h_2 of this figure are shown in Figure 6.10. In this example, we will see how the patches of $h_2 = h_1\mathcal{T}s_1$ can be derived from the patches of h_1. h_1 has three faces; two planar faces f_1 and f_2 and a cylindrical face f_3. The face f_1 has a single loop-patch l_1 labeled UNBLOCKED. The face f_2 also has a single loop-patch l_2 labeled UNBLOCKED. The face f_3 has three loop-patches l_3, l_4 and l_5. Of these, l_4 is labeled BLOCKED, and l_3 and l_5 are labeled UNBLOCKED. A step by step trace of the algorithm is given below.

Step 1 The faces of h_2, viz., f_4, f_5 and f_6 are identified.

Step 2 Face f_4 is selected. We determine f_4 to be a planar face.

Step 3 There is no face of h_1 that is on the same plane as f_4. Therefore, we create a single loop-patch l_6 and assign it to the 'loops' field if f_4. The label of l_6 is UNBLOCKED.

Step 2 Face f_5 is selected. We determine f_5 to be a planar face.

Step 3 The test fails.

Step 4 The face f_5 of h_2 is on the same plane as the face f_2 of h_1. Therefore, f_2 is selected. f_2 has only one loop-patch, l_2.

Step 5 l_2 is totally contained in f_5. Thus, the 'loops' field of f_5 has exactly one loop-patch, viz., l_2.

Step 2 The face f_6 is selected. We determine f_6 to be a cylindrical face.

Step 3 The test fails.

Step 4 The face f_6 of h_2 is on the same infinite cylinder as the face f_3 of h_1. Therefore, f_3 is selected. f_3 has three loop-patches, l_3, l_4 and l_5.

Step 5 l_3 is determined to be outside f_6.

Step 5 l_4 is determined to be totally contained in f_6. Therefore, l_4 is added to the loop-patches of f_6.

Step 5 l_5 is determined to be totally contained in f_6. Therefore, l_5 is added to the loop-patches of f_6. Thus, f_6 has two loop-patches, l_4 and l_5.

6.8.2 Maximal Extension

Determining the patches of $x\mathcal{M}_f y$ is similar to determining the patches of $x\mathcal{T}y$. We first compute the faces of $x\mathcal{M}_f y$. In computing the patches and patch labels of $x\mathcal{T}y$, we compared the patches of x with the faces of $x\mathcal{T}y$. In the case of $x\mathcal{M}_f y$, we compare the faces of $x\mathcal{M}_f y$ with the patches of x and y. The following is an algorithm for determining the patches of $x\mathcal{M}_f y$.

Step 1 After $\mathrm{ps}(x\mathcal{M}_f y)$ has been computed, the faces of $x\mathcal{M}_f y$ are known. Repeat the Steps 2 to 8 for each face of $x\mathcal{M}_f y$.

Step 2 Let f be the face under consideration. For each face we have a face-patch. Let the face-patch of f be f_p. The 'label' field of the face-patch can easily by determined depending on whether the face is planar or cylindrical. Each face-patch comprises of several loop-patches given by the 'loops' field.

Step 3 If there does not exist a face of x that is on the same plane or the same infinite cylinder as f, then go to Step 6.

Step 4 Let f' be the face of x chosen in Step 3. Repeat Step 5 for each loop-patch of f'.

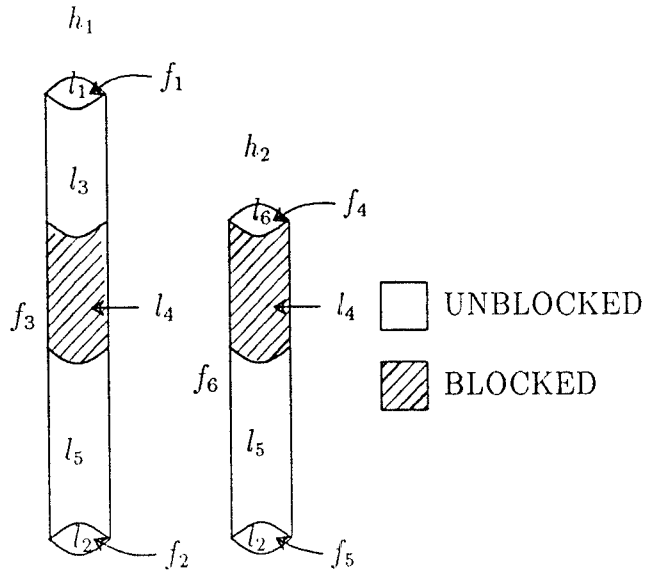


Figure 6.10: Propagation of the patches in truncation.

Step 5 Let l' be the loop-patch under consideration. If the interior of l' is totally contained in f_p, add l' to the 'loops' field of f_p. If the interior of l' is partially contained in f, then split the loop-patch l' into two sub-patches such that one of them is totally contained in f_p and the other is outside f_p. However, both the sub-patches have the same 'loop-type' and 'label' fields as l'. Let the sub-patch that is totally contained in f_p be l'_{in}. Add l'_{in} to the 'loops' field of the face-patch of f. Set f_p to $f_p -_2^* l'$, where $-_2^*$ denotes the regularized subtraction in two dimensions. (If the interior of l' is outside the face f do nothing.)

Step 6 If there does not exist a face of ps(y) that is on the same plane or the same infinite cylinder as f then consider the entire face-patch as a single loop; a rectangular loop, or a circular loop or a cylindrical loop. Mark its 'loop-type' field accordingly. Mark the 'label' of the loop-patch as as UNBLOCKED. Go to Step 8.

Step 7 Repeat Steps 4 and 5 with all the occurrences of the feature x replaced by the feature y.

Step 8 Consider the portion of f that is not covered by the existing loop-patches for f. Label it as UNBLOCKED. Add this to the 'loops' field of the face-patch of f.

Chapter 7

Illustrative Examples

yathā, saumya, ekena nakha-nikṛntanena sarvaṁ kārṣṇāyasaṁ vijñātaṁ syāt, vācārambhaṇaṁ vikāro nāma-dheyaṁ kṛṣṇāyasam ity eva satyam, evam, saumya, sa ādeśo bhavtīti.
(Just as, my dear, by one pair nail scissors all that is made of iron becomes known, the modification being only a name arising from speech while the truth is that it is just iron: thus, my dear, is that teaching.)

—*Chāndogya Upaniṣad.*

In this chapter, the feature algebra and the generate-features algorithm will be illustrated through some examples.

7.1 Example 1

The first example is the same as the one discussed in Figure 1.1. The part is reproduced in Figure 7.1. In this example, the part has been described as the part resulting from subtracting a hole h_1, a slot s_1 and a slot s_2, in that order out of a rectangular stock. Because this is a simple example, we will be able to study various aspects of the feature algebra in detail. The parameters of the stock and the features h_1, s_1 and s_2 are shown in Tables 7.1, 7.2, 7.4 and 7.6 respectively. The parameters used in process planning for the features h_1, s_1 and s_2 are shown in Tables 7.3, 7.5 and 7.7 respectively. All the dimensions of length are in inches.

Field	Value
Corner	⟨ 0.0, 0.0, 0.0 ⟩
Dimensions	(2.0 2.0 3.0)

Table 7.1: The parameters of the stock for Example 1.

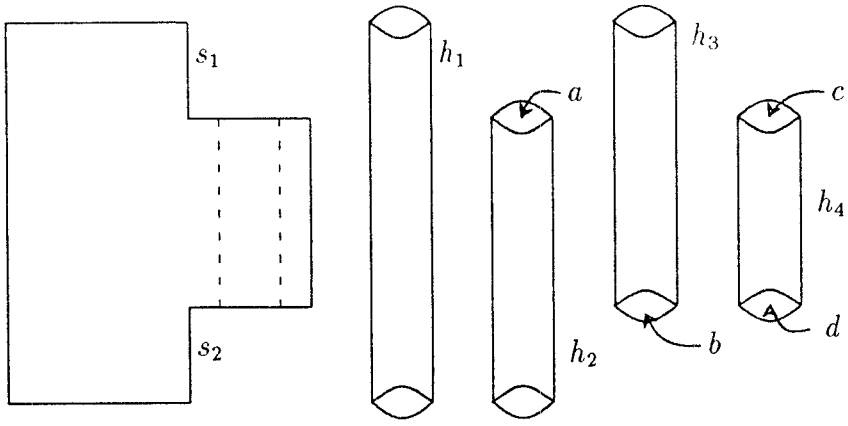


Figure 7.1: A part with two slots and a hole.

Field	**Value**
Height	3.0
Radius	0.225
Major-axis	:Z
Center	$\langle$ 1.0, 1.65. 0.0 $\rangle$

Table 7.2: The parameters of the hole h_1 in Example 1.

Field	**Value**
Type	Hole
Bottom	Open
Hole-Quantity	50
Depth	3.0
Diameter	0.45
True-Position	0.02
Negative-Tolerance	0.0002
Positive-Tolerance	0.00025
Special-Feature	None
Roundness	0.1
Surface-Finish	125

Table 7.3: The process planning parameters of the hole h_1 in Example 1.

Field	**Value**
Loop	($\langle$ 0.0, 1.3, 3.0$\rangle$, $\langle$ 2.0, 1.3, 3.0$\rangle$, $\langle$ 2.0, 2.0, 3.0$\rangle$, $\langle$ 0.0, 2.0, 3.0$\rangle$)
Sweep	-0.74$\vec{k}$

Table 7.4: The parameters of the slot s_1 in Example 1.

Field	Value
Flatness	0.1
Negative-Tolerance	0.01
Positive-Tolerance	0.01
Surface-Finish	100

Table 7.5: The process planning parameters of the slot s_1 in Example 1.

Field	Value
Loop	($\langle$ 0.0, 1.3, 0.74$\rangle$, $\langle$ 2.0, 1.3, 0.74$\rangle$, $\langle$ 2.0, 2.0, 0.74$\rangle$, $\langle$ 0.0, 2.0, 0.74$\rangle$)
Sweep	-0.74$\vec{k}$

Table 7.6: The parameters of the slot s_2 in Example 1.

Field	Value
Flatness	0.1
Negative-Tolerance	0.01
Positive-Tolerance	0.01
Surface-Finish	100

Table 7.7: The process planning parameters of the slot s_2 in Example 1.

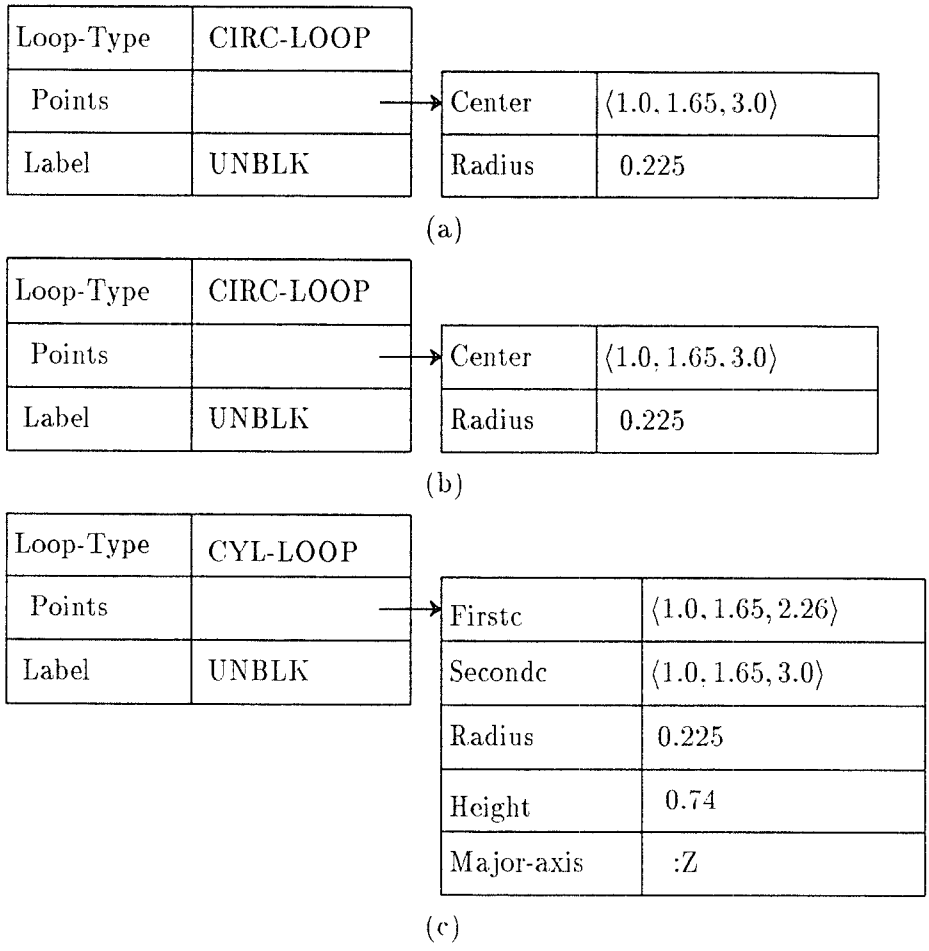


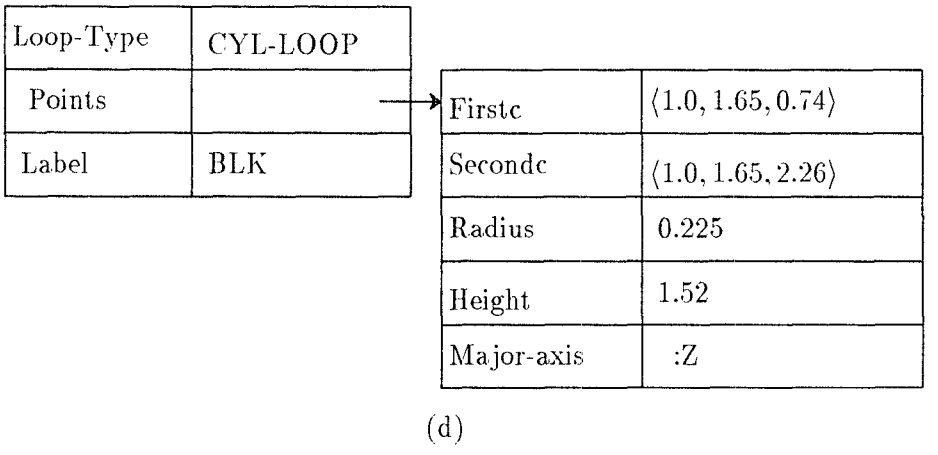
Figure 7.2: The fields of the loop-patches l_1, l_2, l_3, l_4 and l_5.

7.1.1 Patches

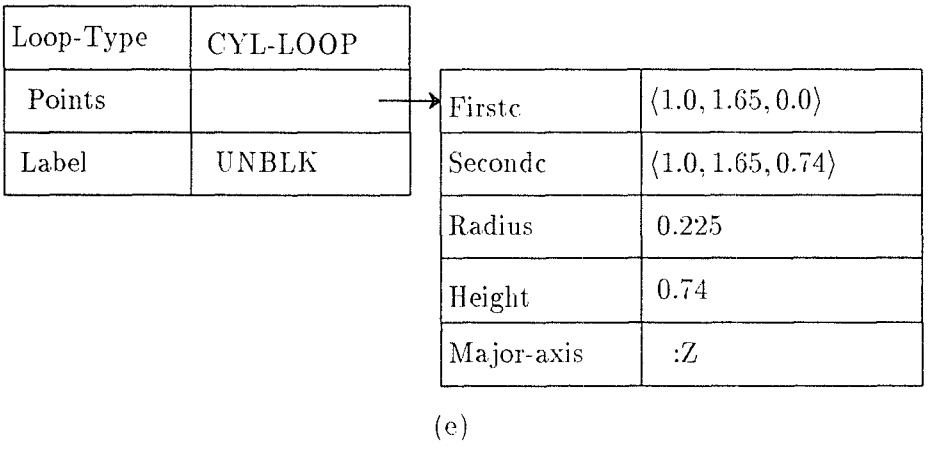
Even for this simple example, the patches and patch label propagation cannot be described in detail. Therefore, we will describe the patches of the feature h_1 only. The 'so-patches' field of the feature h_1 has three face-patches. Let us call them f_1, f_2 and f_3 (see Figure 6.10). f_1 and f_2 correspond to the planar faces of h_1 and f_3 corresponds to the cylindrical face of h_1. Therefore, the 'label' fields of f_1, f_2 and f_3 are PLANAR, PLANAR and CYLINDRICAL respectively. The 'loops' field of f_1 consists of a single loop-patch l_1. The 'loops' field of f_2 consists of a single loop-patch l_2. The 'loops' field of f_3 consists of three loop-patches l_3, l_4 and l_5. The fields of the loop-patches l_1 through l_5 are shown in Figures 7.2 (a) through (e) respectively.

7.1.2 Generate-Features Algorithm

Let us now illustrate how the workings of the generates-features algorithm for this example. Initially, there is only one feature interpretation of the part, viz., $\{h_1, s_1, s_2\}$. Thus, before the beginning of



(d)



(e)

Figure 7.2 The fields of the loop-patches l_1, l_2, l_3, l_4 and l_5 (Continued).

the first iteration, we have

$$\mathrm{NI} = \{\{h_1, s_1, s_2\}\}$$

and

$$\bar{F} = \{\{h_1, s_1, s_2\}\}.$$

In the first iteration of the generate-features algorithm, we consider the interactions among the features in the feature interpretation $\{h_1, s_1, s_2\}$. Given two features x and y, (see the explanation given in Section 4.1.3), we have seven possibilities for the operation η in $x\eta y$ if x is a rectangular solid and three possibilities for η if x is a cylinder. In our experiments, we measured the number of times we test the applicability of an operation on a pair of features, and the number of times the result of an operation is a valid feature. We will refer to the former as *applicability tests* and the latter as *feature interactions*. Therefore, given the ordered pairs of features $\langle h_1, s_1\rangle$, $\langle h_1, s_2\rangle$, $\langle s_1, h_1\rangle$, $\langle s_1, s_2\rangle$, $\langle s_2, h_1\rangle$ and $\langle s_2, s_1\rangle$, we have a total of $3+3+7+7+7+7 = 34$ applicability tests. As a result of the feature interactions, we generate two new features $h_2 = h_1 \mathcal{T} s_1$ and $h_3 = h_1 \mathcal{T} s_2$, and two new feature interpretations $\{h_2, s_1, s_2\}$ and $\{h_3, s_1, s_2\}$. Thus, the number of feature interactions in this iteration is 2.

Thus, before the beginning of the second iteration, we have

$$\mathrm{NI} = \{\{h_2, s_1, s_2\}, \{h_3, s_1, s_2\}\}$$

and

$$\bar{F} = \{\{h_1, s_1, s_2\}, \{h_2, s_1, s_2\}, \{h_3, s_1, s_2\}\}.$$

In the second iteration of the generate-features algorithm, we consider the interactions among the features in the feature interpretations $\{h_2, s_1, s_2\}$ and $\{h_3, s_1, s_2\}$. The total number of applicability tests in this iteration is $2 * 34 = 68$. As a result of the feature interactions one new feature $h_4 = h_2 \mathcal{T} s_2 = h_3 \mathcal{T} s_1$ is generated. One new feature interpretation, $\{h_4, s_1, s_2\}$ is also produced. There are two other interactions that result in already existing features. They are: $h_2 \mathcal{M}_a s_1 = h_1$ and $h_3 \mathcal{M}_b s_2 = h_1$. Thus, the feature interactions in this iteration are $h_2 \mathcal{T} s_2$, $h_3 \mathcal{T} s_1$, $h_2 \mathcal{M}_a s_1$ and $h_3 \mathcal{M}_b s_2$. Therefore, the total number of interactions in this iteration is 4.

Thus, before the beginning of the third iteration, we have

$$\mathrm{NI} = \{\{h_4, s_1, s_2\}\}$$

and

$$\bar{F} = \{\{h_1, s_1, s_2\}, \{h_2, s_1, s_2\}, \{h_3, s_1, s_2\}, \{h_4, s_1, s_2\}\}.$$

In the third iteration, we consider the interactions among the features in the feature interpretation $\{h_4, s_1, s_2\}$. The number of applicability tests considered in this iteration is 34. However, no new features or feature interpretations are produced in this iteration. As in the previous iteration, two interactions result in already existing features. They are: $h_4 \mathcal{M}_c s_1 = h_3$ and $h_4 \mathcal{M}_d s_2 = h_2$. Therefore, the number of interactions in this iteration is 2.

Thus, before the beginning of the fourth iteration, we have

$$\mathrm{NI} = \emptyset$$

and

$$\bar{F} = \{\{h_1, s_1, s_2\}, \{h_2, s_1, s_2\}, \{h_3, s_1, s_2\}, \{h_4, s_1, s_2\}\}.$$

Since $\mathrm{NI} = \emptyset$, the algorithm terminates after three iterations.

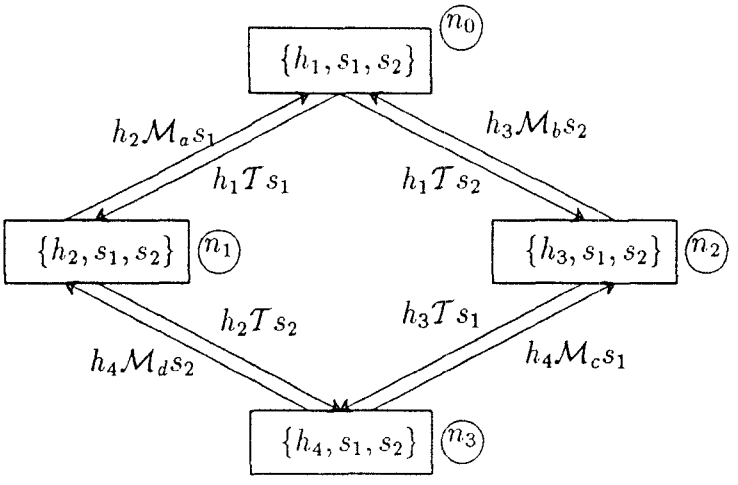


Figure 7.3: The state-space for Example 1.

7.1.3 The State-Space

In Section 6.7.1, we showed how the generate-features algorithm produces the state-space of all the feature interpretations. In Figure 7.3 we show the state-space for Example 1. The state-space is also shown pictorially in Figure 7.4. The generate-features algorithm produces only the states and the relationships between the states. The explicit state-space is not produced by generate-features. Notice that, generate-features produces the state-space in a breadth-first [Win84] manner.

The state-space graph has four nodes, one for each feature interpretation. The nodes are marked n_0, n_1, n_2 and n_3. Notice the labeled directed edges between states that show how one state is related to another.

7.1.4 Monitoring

The generate-features algorithm produces information regarding its functioning at various levels of detail, as specified by the user. On most occasions, we want to know the CPU time, the number of interactions, etc. The output produced by generate-features for Example 1 is shown in Table 7.8.

From Table 7.8, we see that the total number of applicability tests is 136. This can easily be verified analytically. We had 34, 68 and 34 applicability tests for the three iterations. Therefore, the total number of applicability tests is $34 + 68 + 34 = 136$. We had 2, 4 and 2 feature interactions for the three iterations. Therefore, the total number of feature interactions is $2 + 4 + 2 = 8$.

We notice that, in a number of instances the same applicability test can be repeated. For example, the applicability test for $s_1 \mathcal{T} s_2$ occurs in the states n_0, n_1, n_2 and n_3. Since we are using a lookup table to store the results of the applicability tests, one does not have to repeat any operation with the same operands more than once. From Table 7.8, we find that the number of repetitions of the applicability tests is 42. This can be verified analytically. The only applicability tests that repeat in Example 1 are of the form $s_1 \eta s_2$ or $s_2 \eta s_1$, where η is one of the seven possible operations. Since these applicability tests occur for the first time in n_0, their occurrences in n_1, n_2 and n_3 are repetitions. Thus, there are totally $2 * 7 * 3 = 42$ repetitions. From this, it follows that the number of distinct applicability tests is $136 - 42 = 94$.

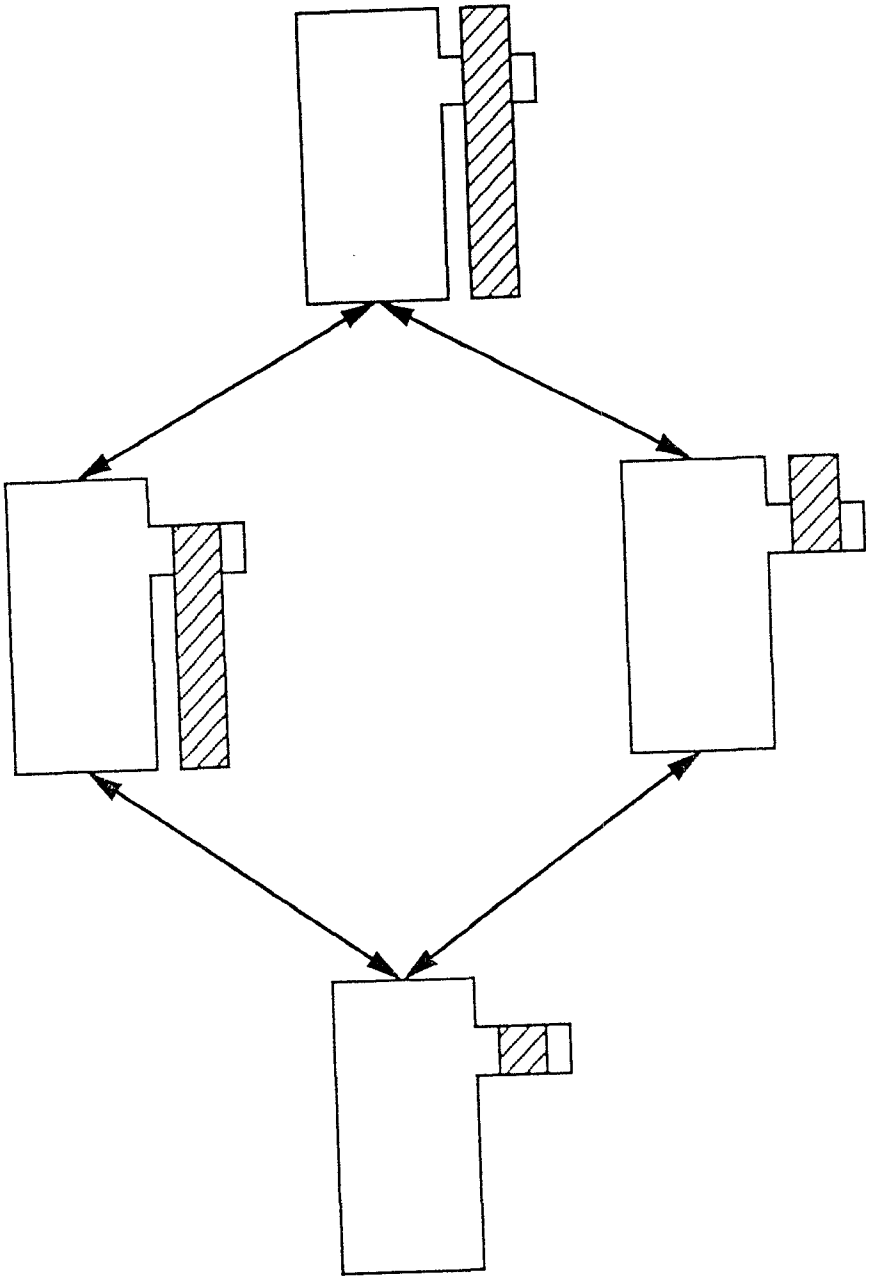


Figure 7.4: The state-space for Example 1.

Statistic	Value
Total applicability tests	136
Repetitions	42
Total Interactions	8
Time (without algebraic properties)	2.68s
Time (using algebraic properties)	6.35s

Table 7.8: The statistics for Example 1.

7.1.5 Algebraic Properties

The algebraic properties can be used in some cases, to determine the result of an operation, without having to compute it using the procedures. In this example, 29 applicability tests were *resolved* using the algebraic properties. By this we mean that the procedures for the operations were not invoked for 29 applicability tests. In other words, the resolution-based theorem-prover of Prolog succeeded in determining the result of 29 applicability tests. The total CPU time without access to the algebraic properties is 2.68 seconds; where as, with access to the algebraic properties the CPU time increased to 6.35 seconds. However, the original intent in providing access to the algebraic properties was that it would decrease the CPU time. We observed similar increases in the timing when the algebraic properties were used, in all the examples we have tried. Experimentally monitoring the timings, we found that most of the delay was caused in accessing the Prolog system for queries or for assertions. One reason for this result could be that the Lisp code is compiled and the Prolog code is interpreted. Another reason could be that since the features and interactions considered right now are rather simple, the procedures for the operations take less time than the procedures necessary to compute the algebraic properties. With more complex features and interactions, access to the algebraic properties may turn out to be more useful.

7.1.6 Efficiency

The procedures for computing the operations take advantage of the nature of the features and interactions. Declaratively, the operations for this sub-algebra can be expressed in terms of set operations on solids. Therefore, another way to compute the operations would be to convert them into set operations on solids (computed by the solid modeler) and then test if the result of the operation is a new feature. (This method has been discussed in Sections 4.1.2 and 4.1.4.) Using this method, the time taken for Example 1 is 140.9 seconds.

Using this method, we have not been able to run any examples involving more than 150 applicability tests (approximately). The TI-Explorer II, on which we ran the program, ran out of memory for larger examples after executing for 4 to 5 minutes. On smaller examples, we have found that the time taken by this method is about 50 times more than using the algebra (without access to the algebraic properties). This shows that using the algebra is much more efficient than translating the operations into equivalent set operations. The reason is that, if the result of an operation is not a valid feature, there is no reason to compute the shape of the solid and then realize it is not a valid feature. The algorithms in the algebra recognize the cases that result in invalid features efficiently. Since only a small fraction of the total number of applicability tests result in valid features these algorithms have improved the performance of the algebra significantly.

Feature	Process Plan	Cost
h_1	Gun-Drill→ Rough-Ream→ Finish-Ream	25.0
s_1	Rough-Peripheral-Mill	1.5
s_2	Rough-Peripheral-Mill	1.5
h_2	Twist-Drill→ Rough-Ream→ Finish-Ream	11.0
h_3	Twist-Drill→ Rough-Ream→ Finish-Ream	11.0
h_4	Twist-Drill→ Rough-Bore→ Finish-Bore	9.0

Table 7.9: The process plans produced by EFHA for the features in Example 1.

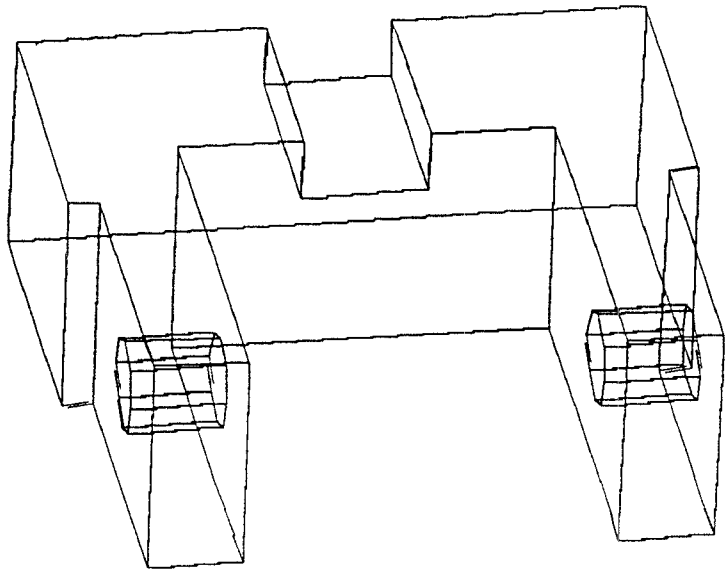


Figure 7.5: The wire frame model of a bracket for use with a bearing. This image was produced using the feature algebra interface and the Protosolid solid modeler.

7.1.7 Process Planning

After the generate-features algorithm has terminated, we also have the information about the features in a format suitable for EFHA to do process planning. Table 7.9 shows the cheapest process plans produced by EFHA for the features in Example 1.

7.2 Example 2

This example is same as the one shown in Figure 1.2. A wire frame model of the same is shown in Figure 7.5. A detail of a portion of this object is shown in Figure 7.6. As described by the designer, the part has slots s_1, s_2, s_3 and s_4 and holes h_1 and h_2. As discussed in Section 1.1, there are several possible feature interpretations for this part.

The feature algebra produces several feature interpretations of this part including all the ones mentioned in Figure 1.4. For the example shown in Figure 1.2, the total CPU time for computing the alternative feature interpretations without access to the algebraic properties was 46.2 seconds. In this example, the total number of distinct applicability tests was 834. The total number of

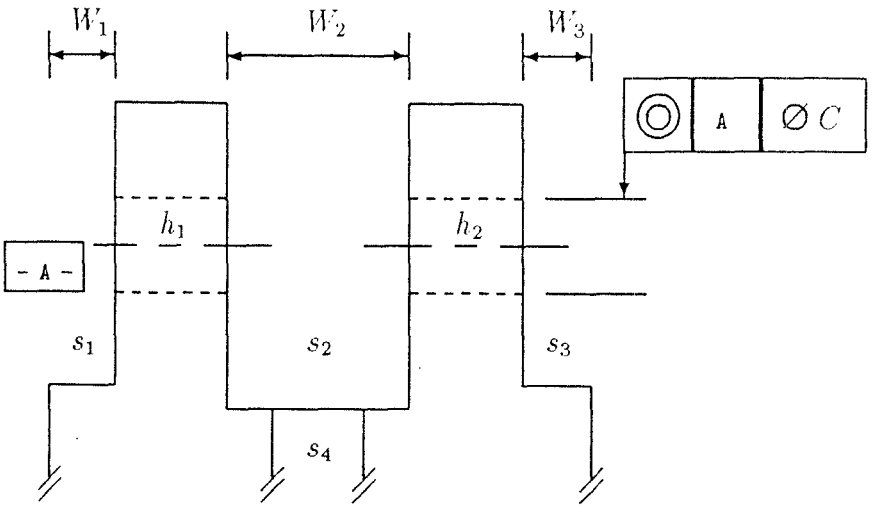


Figure 7.6: A portion of the bracket shown in Figure 7.5. W_1, W_2, and W_3 are the widths of s_1, s_2, and s_3, respectively, and C is the concentricity tolerance for the holes h_1 and h_2.

feature interactions was 272. The number of additional features generated in this process is only 11. The total number of feature interpretations generated is 40. The 40 feature interpretations can be partitioned into two groups of 20 interpretations each. These two groups are identical, except that, 20 of them have the feature s_4 and 20 of them have the extension of s_4 into s_2. For the sake of simplicity, we show only one of the groups in the state-space shown in Figure 7.7. This figure does not show either s_4 or the extension of s_4 into s_2. The arcs between feature interpretations indicate that one feature interpretation can be derived from the other. Most of the arcs are bidirectional. The arc labels have been omitted for the sake of simplicity. With access to the algebraic properties, the CPU time to compute the alternative feature interpretations increased to 86.2 seconds. The algebraic properties were useful in resolving 212 applicability tests.

7.3 Example 3

In Section 6.7.2 we discussed an example having a slab composed of n contiguous slots subtracted from a piece of stock. In this section, we illustrate this example for $n = 4$.

In this example, a part has been created by the designer by successively subtracting four slots, s_1, s_2, s_3 and s_4 from a piece of stock. This process is illustrated in Figures 7.8 (a) through (e). Even though it is very unlikely that someone would design a part in this manner: this example is interesting because it has an exponential number of feature interpretations. (See Section 6.7.2.)

The faces of the slots s_1, s_2, s_3 and s_4 that are parallel to the XZ plane are named a through h as shown in Figure 7.8 (f). The generate-features algorithm produced eight feature interpretations of the part. This fits well quite well with analytic calculations, since the number of feature interpretations is supposed to be $2^{4-1} = 8$. Generate-features also produced six additional features. These are shown in Figure 7.9. For this example, the number of distinct applicability tests was 210. The number of feature interactions was 24. The reader is encouraged to verify analytically that these are correct. The CPU time to compute all the feature interpretations without access to the algebraic properties was 4.92 seconds and with access to the algebraic properties it was 16.65 seconds. The number of applicability tests resolved using the algebraic properties was 68.

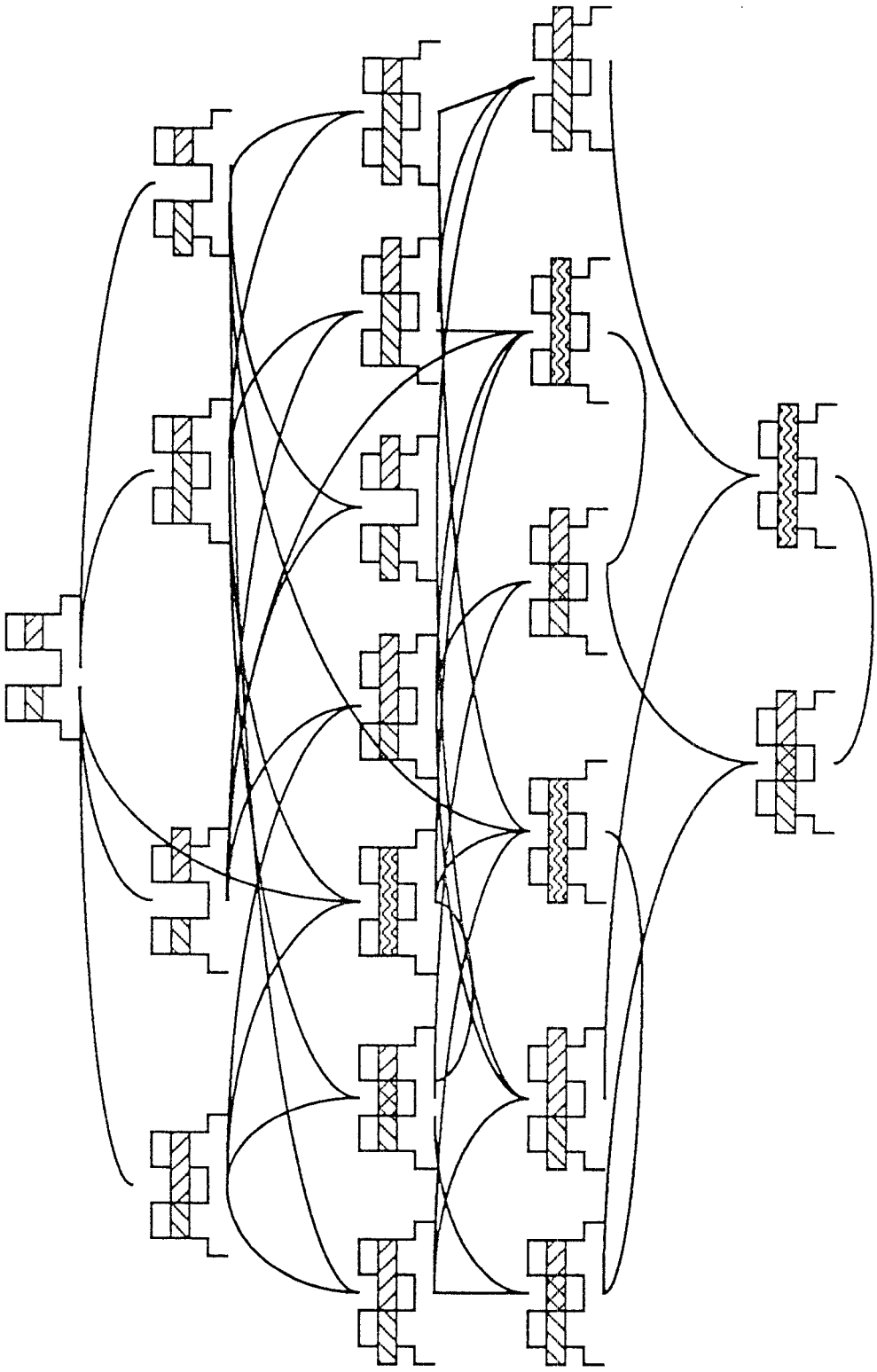


Figure 7.7: The state-space for Example 2.

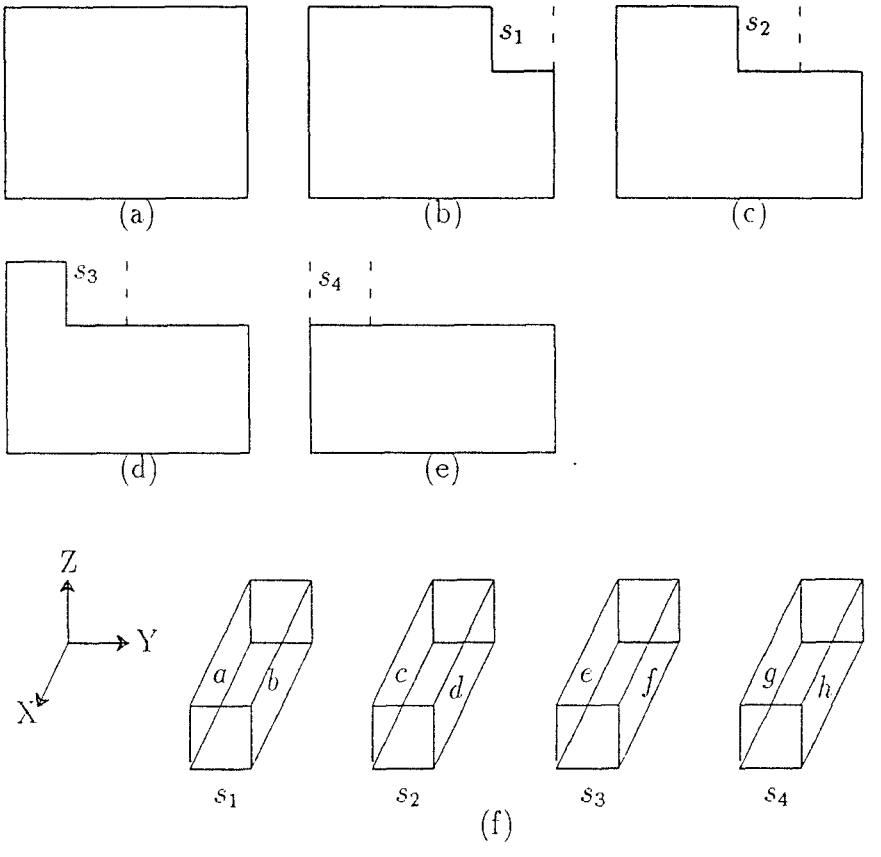


Figure 7.8: A part with four contiguous slots.

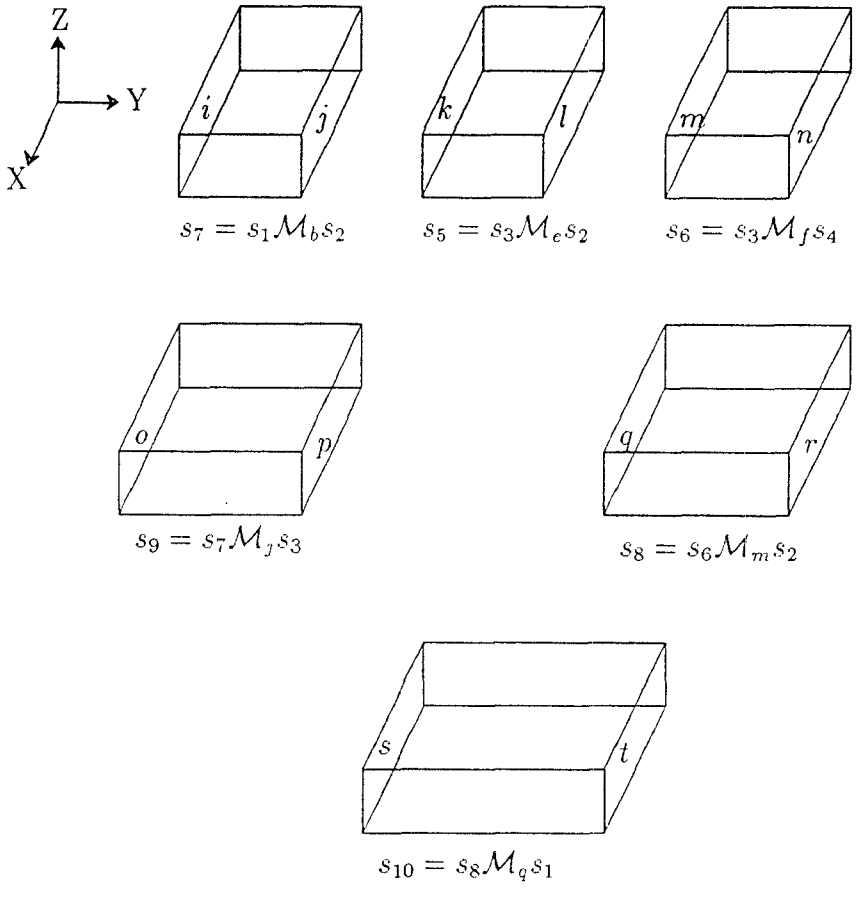


Figure 7.9: The additional features produced by generate-features.

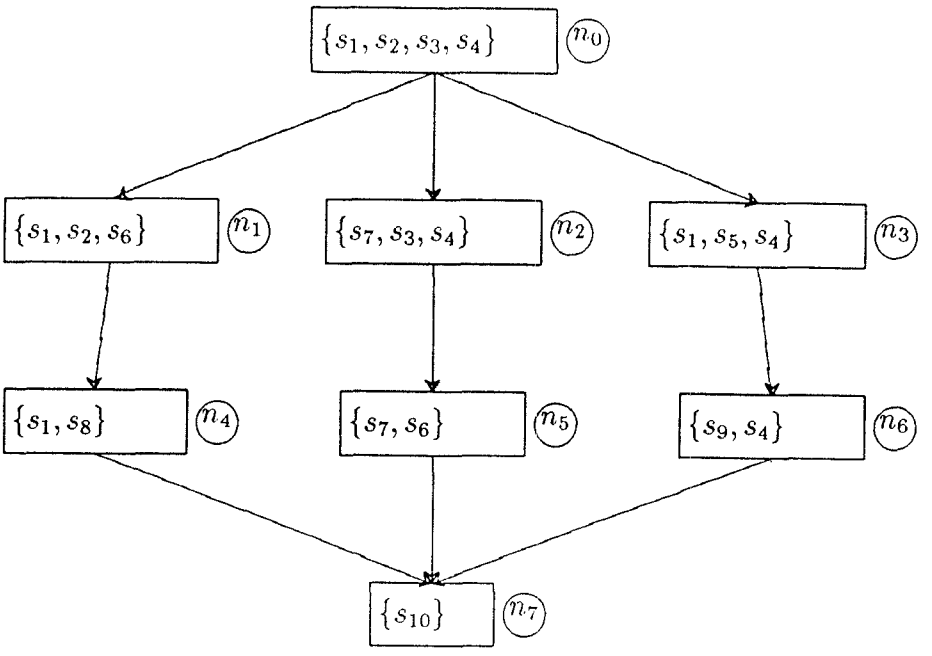


Figure 7.10: The state-space for Example 3.

The state-space for this example is shown in Figure 7.10. The state-space graph has a total of eight nodes marked n_0, n_1, ...n_7. In this state-space, we have more than one labeled edge between two nodes. For example, we can have an edge labeled $s_1 \mathcal{M}_b s_2$ and an edge labeled $s_2 \mathcal{M}_c s_1$ from n_0 to n_2. For the sake of simplicity, we have omitted all the edge labels in Figure 7.10. Further, in Figure 7.10, multiple directed edges from a node n_i to a node n_j are replaced by a single directed edge.

In this example, the process engineer would have combined all the contiguous slots into one. Whereas, the feature algebra produces eight different feature interpretations even though most of them are not useful from a manufacturing perspective. A good way to handle these kind of situations is to couple the feature algebra with some rules or heuristics the prevent redundant feature interpretations from being generated.

Chapter 8

Conclusion

I can live with doubt and uncertainty. I think it's much more interesting to live not knowing than to have answers which might be wrong.
—Richard Feynmann, 1981

This chapter summarizes the algebraic approach for handling feature interactions that has been proposed in this thesis. Extensions to the feature algebra and possibilities for further research are also discussed.

8.1 Summary

This thesis began by identifying the role of geometric feature interactions in process planning. Feature interactions result in multiple feature interpretations of a machinable part. The feature algebra has been proposed as a mechanism for producing multiple feature interpretations. The algebra covers practically all features of interest to manufacturing. The mathematical framework of the feature algebra which includes the domain of features, the operations and the properties of the operations has been described. In addition, a methodology for characterizing additional information about the boundary of a feature in the form of patches and patch labels has been described. A sub-algebra of the generalized feature algebra has been discussed at a greater length. The data structures for representing the features in this sub-algebra, and the procedures for the operations on features have been provided. Details of the implementation of the sub-algebra and its integration with the Protosolid [Van89b] solid modeler and the EFHA [Tho89] process planner have also been discussed. The performance and the computational complexity issues of the sub-algebra have been discussed. It has been shown through experiments that computing the operations using the procedures developed for the sub-algebra is much more efficient than computing them by converting them to queries to a solid modeler.

The main contributions of this research are the following:

1. A methodology by which a machinable part can be considered as several alternative sets of features has been developed. Popular approaches towards the integration of computer aided

design and computer aided manufacturing have not addressed the problem of multiple feature interpretations[1].

2. The methodology that has been developed, viz., the algebra of features is general enough to include practically all shapes of interest to manufacturing.

3. A mathematical characterization of features and feature interactions has been developed. Currently, most of the approaches towards addressing feature interactions are based on expert rules.

4. A feature, in our representation not only includes its nominal geometry, but also additional information that identifies the portions boundary of the feature that belong to the boundary of the final part and the portions that do not. This information will be useful in process planning to identify the portions of the boundary and the volume of a feature that are required and the portions that are optional.

5. A prototype of the feature algebra has been implemented for a restricted subset of features and has been integrated with a solid modeler and a process planning system. The integration with the solid modeler has some advantages. First, one is guaranteed to have a complete and unambiguous representation [Mor85] of a solid at all times. Second, a geometric modeler maintains the topological and geometric relationships within a solid and provides the user restricted access to the relationships. Without integration with a solid modeler, analyzing feature interactions becomes cumbersome. Further, there is always the potential for creating pathological objects that are not solids.

6. For the prototype that has been implemented, the procedures for computing the operations take advantage of the nature of the features and interactions. Declaratively, the operations in this prototype can be expressed in terms of set operations on solids. Therefore, another way to compute the operations would be to convert them into set operations on solids (computed by the solid modeler) and then test if the result of the operation is a feature. By comparing with this method, we have shown that the procedures for computing the operations are about 50 times faster than the ones that use set operations on solids.

8.2 Impact

In Section 1.1 and later in Chapter 2 we discussed the role of feature interactions, some of it as viewed by other researchers. Below, we present the feature interactions given in Section 2.3.9 (titled 'Interactions between features') of a report by CAM-I[2] [SRSM89] and discuss them in the context of the feature algebra. This will enable us understand the scope, the relevance and the shortcomings of the feature algebra in the backdrop of CIM research in the industry. In the following paragraphs, the statements taken from the CAM-I report are shown in *emphasized* mode. The figures and figure captions used here have been copied from the CAM-I report.

1. *In Figure 8.1 is a hole too close to the outside surface and so it breaks through.* The question is as to how we detect this interaction. The goal is not to disallow such an interaction, but to detect it and inform the designer of the interaction. One way the feature algebra can detect

[1]The program developed by Jan Vandenbrande[Van90] produces multiple feature interpretations in certain cases.

[2]Computer Aided Manufacturing-International Inc. (CAM-I), is a consortium of organizations pioneering in CIM research.

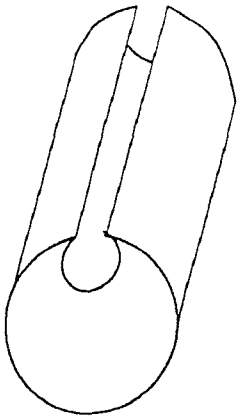


Figure 8.1: Feature is not functional.

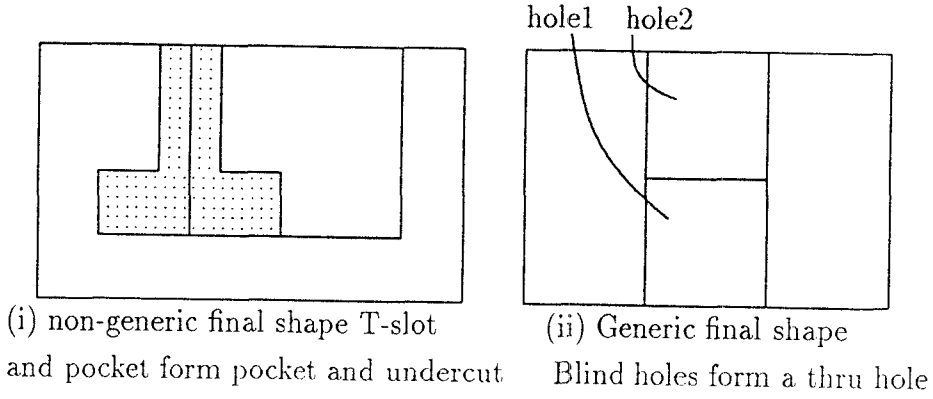


Figure 8.2: Non-generic shape from two generic shapes.

this interaction is by using the patch labels. However, in the restricted feature algebra that has been implemented, the patches can only be cylindrical bands but not sectors of a cylindrical surface (see the discussion at the end of Section 5.5). If we extend the representation to have such patches, then we will be able detect the interaction shown in Figure 8.1 easily.

2. *In Figure 8.2, two legitimate generic features intersect and produce a non-generic shape in part (i) and a generic shape in part (ii). In part (i) T-slot and pocket form pocket and undercut. In part (ii)* (two) *blind holes form a thru hole.* One can easily see that the maximal extension operation handles both of these interactions and provides the alternative feature interpretations with appropriate patch labels (to identify undercuts and through holes).

3. *In Figure 8.3, the creation of the pocket has eliminated the entry face of the hole; the depth of the hole has changed even though the user did not modify the hole parameters.* This interaction can be handled by the truncation operation.

4. *In Figure 8.4 two slots intersect, thus creating new topological entities.* Due to this interaction, additional topological entities which are not present in either of the slots are present in the final part shape. This kind of situation will be a problem if features are required to satisfy some topological constraints with respect to the final part. In other words, if the features are

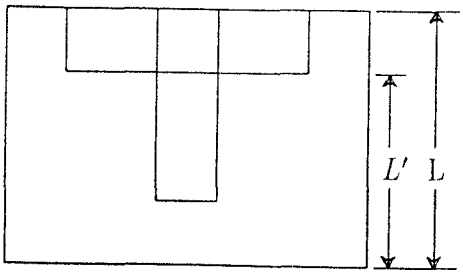


Figure 8.3: Feature parameters made obsolete.

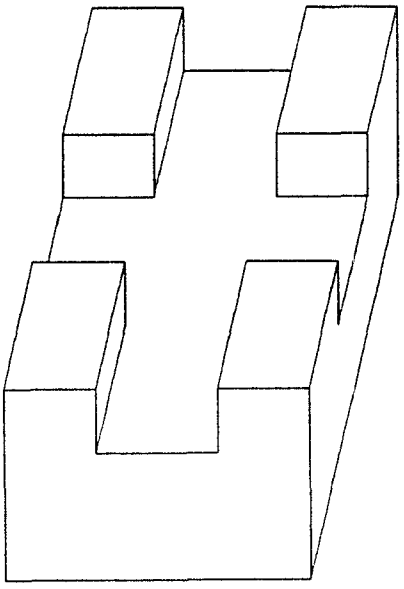


Figure 8.4: Non-standard topology from interaction.

required to satisfy a fixed topology in the final part, then this interaction will create problems.

At this time, there is no consensus among researchers as to whether features should have a fixed topology. Restricting features to a fixed topology precludes many interesting interactions. However, some topological restrictions should be met by the features. For example, at least some portion of the boundary of a feature must belong to the final part; otherwise, such a feature will be redundant.

For this interaction, using the feature algebra, we can obtain additional interpretations of the part and some of them will not create additional topological entities.

5. *In Figure 8.5, a new feature has completely deleted an old feature from the geometric model.* This interaction is handled by the subsumption relation discussed in this thesis.

6. *In Figure 8.6, a new feature has closed off a feature that was originally open. In Figure 8.7, the addition of a volume deletes a cavity.* These two interactions are interesting because they occur between additive and subtractive features. Currently, the feature algebra is not capable of handling interactions between additive and subtractive features. At this time, it is not clear how the current feature algebra can be extended to handle such interactions. However, since

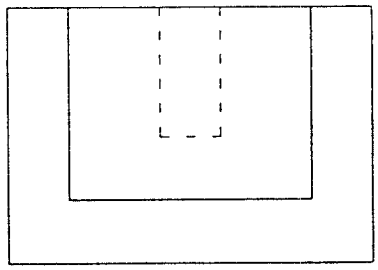


Figure 8.5: Feature deleted by a larger feature.

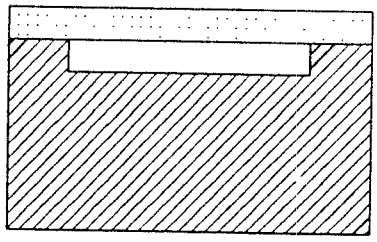


Figure 8.6: Open feature becomes closed.

all operations in machining involve metal removal, a good approach may be to let the designer use both additive and subtractive features during the design phase, and then translate them to a set of manufacturing features using a feature extraction system. After this step, the feature algebra can be used to analyze the interactions among the manufacturing features.

7. *In Figure 8.8, a groove placed on a tube is deep enough to make the part disjoint.* This interaction can be handled by the truncation operation if we use the current part (the hollow tube in this case) as one of the operands.

8. *In Figure 8.9, the variation of the length of the two rods independently can result in various interference conditions. The top rod may interfere with the left block and be shortened by it or it may become tangent without an adjacency relationship being recorded.* This kind of feature interaction seems more like a functional constraint, (that the two rods be of equal length) to be imposed during design. While handling such functional constraints is important, it is outside the scope of the feature algebra.

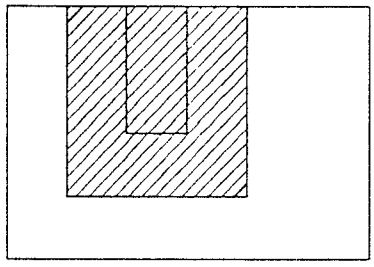


Figure 8.7: Feature deleted by filling with larger feature.

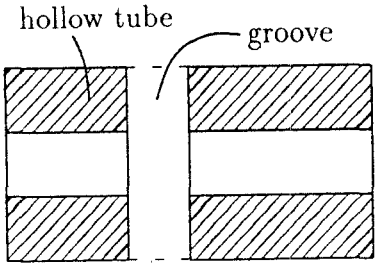


Figure 8.8: Feature makes object disjoint.

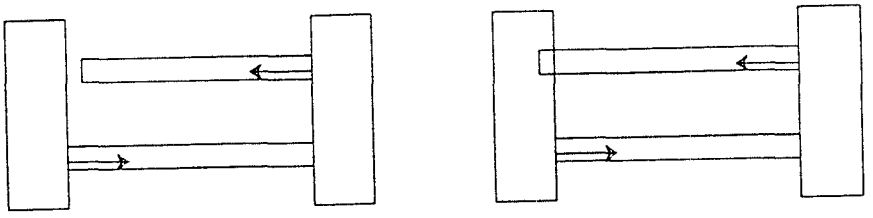


Figure 8.9: Inadvertent interactions from modification.

8.3 Machinability Analysis

Recently, some researchers [ZH90b, ZH90a] have developed techniques for doing machinability analysis while planning for a feature. The system that has been developed takes the parameters of a feature (such as its dimensions and material) and the machining parameters (such as the process, the feed rate and the cutting speed) as input and produces the best achievable tolerances and the best achievable surface quality as output. This system can be used in conjunction with the feature algebra to improve the quality and the productivity in the manufacture of metal parts. Briefly, the goal of such an integrated system is as follows: Given a feature-based description of a part, intelligently choose a combination of the machining parameters and a feature interpretation of the part such that the costs are optimized and the tolerance and surface quality specifications are satisfied. Currently, efforts are being made at the University of Maryland, College Park towards developing such a system.

8.4 Future Work

In this section, we will discuss some of the limitations of the feature algebra, and possible extensions to the theory of the algebra as well as the implementation.

8.4.1 Patches

In Section 5.5, we discussed an example for which the existing representation for patches is not adequate. In this section, we will show another example that requires extensions to the patch representation scheme. The interesting aspect of this example is that the designer will be able to specify all the features and their patches, but problems arise in the propagation of patches to other feature interpretations.

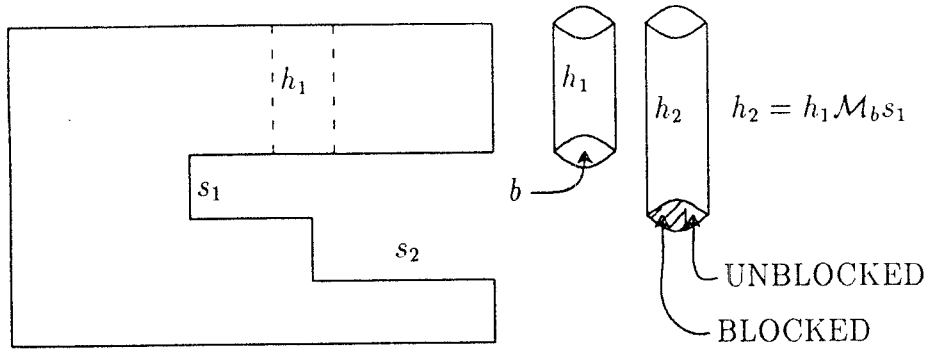


Figure 8.10: An example where the patches of the cylinder h_2 cannot be represented using the current scheme.

In the example shown in Figure 8.10, the patches can be represented in the feature interpretation $\{h_1, s_1, s_2\}$. However, in the feature interpretation $\{h_2, s_1, s_2\}$, where $h_2 = h_1 \mathcal{M}_b s_1$, we will not be able to represent the patches for the bottom face of h_2.

Thus, one possible point for future work is extending the patch representation scheme. This should be done for an extended feature algebra than what has been implemented so far.

8.4.2 The Feature Algebra

Several extensions need to made to the theory of the feature algebra. Some of them are given below:

1. Adding new operations to include some additional kinds of feature interactions. Below, we give a few examples of some new operations that will be useful additions.

   In the part shown in Figure 8.11 the feature $h\mathcal{T}s$ is not a useful feature for manufacturing purposes. It would be more useful to have an operation which would produce the hole $h' = \text{hole-closure}(h -^* s)$ or some extension of h'. We intend to extend the feature algebra to include such operations.

   Often times, it is not just the exact volume of a feature that is relevant but the volume swept by the tool to create the feature. Thus, we need operations that can produce tool-swept volumes.

2. Currently, the feature algebra produces all the feature interpretations. A much better approach is to integrate the feature algebra with experts on tolerancing, process planning, etc. so that the generation of alternative feature interpretations is guided by their advice. This will require that systems with tolerancing and process planning knowledge be capable of geometric reasoning. Process planning and tolerance analysis [RC86, Req83, Tur88, Fle88] are topics of current research.

3. The current implementation of the feature algebra is restricted to rectangular solids, cylinders and countersinks with their planar faces parallel to the faces of the stock. We need to extend this algebra to include the following:

   (a) Pockets, slots, cutouts, notches, etc. with rounded corners;

   (b) T-Slots;

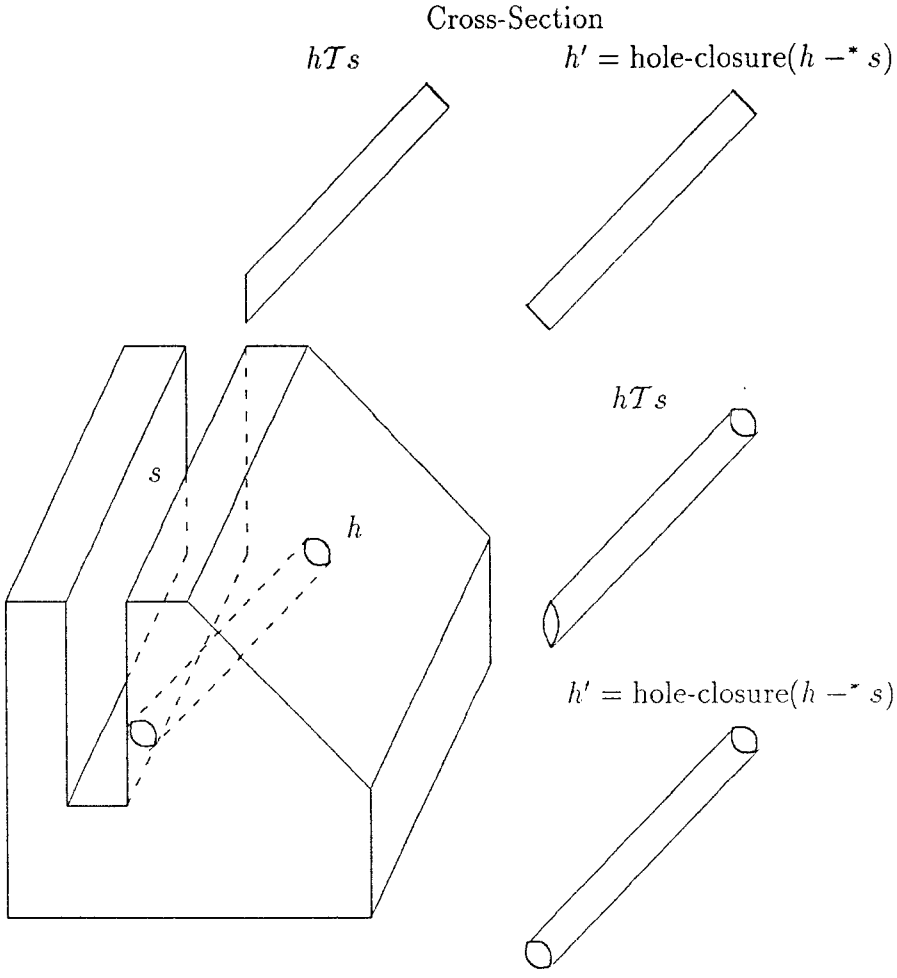


Figure 8.11: The hole-closure operation

(c) Holes with conical tapers;

(d) Threaded holes and other complex features;

(e) Features oriented at angles other than 90 degrees.

The theory of the feature algebra currently handles the above, but developing efficient algorithms for a feature algebra with the above extensions is not an easy task. Developing this algebra should probably be done using a solid modeler that models curved surfaces accurately, unlike Protosolid [Van89b] which uses the faceted approximation.

4. Developing techniques for representation and propagation of tolerances through the feature algebra is another research issue. This will make the feature algebra more useful when integrated with machinability analysis.

Bibliography

[Ago76] M. K. Agoston. *Algebraic Topology: A First Course.* New York: Marcel Dekker, Inc., 1976.

[AGP84] G. T. Armstrong, C. Carey Graham, and Alan De Pennington. Numerical code generation from a geometric modeling system. In Mary S. Pickett and John W. Boyse, editors, *Solid Modeling by Computers: From Theory to Applications.* Plenum Press, 1984.

[BEH79] A. Baer, C. Eastman, and M. Henrion. Geometric modelling: A survey. *Computer-Aided Design Journal*, 11(5):253–272, Sep 1979.

[BH87] S. L. Brooks and K. E. Hummel. Xcut: A rule–based expert system for the automated process planning of machined parts. Technical Report BDX–613–3768, Bendix Kansas City Division, 1987.

[BR87] P. Brown and S. Ray. Research issues in process planning at the national bureau of standards. In *Proceedings of the 19th CIRP International Seminar on Manufacturing Systems*, pages 111–119, June 1987.

[BR88] Peter Brown and Steven Ray. Nbs amrf process planning system-system architecture. Technical Report NISTIR 88-3828, National Institute of Standards and Technology, 1988.

[Bra79] I. C. Braid. Geometric modeling — ten years on. Technical Report CAD Group Document 103, Computer Laboratory, University of Cambridge, England, 1979.

[Cha90] Tien-Chien Chang. *Expert Process Planning for Manufacturing.* Addison-Wesley Publishing Company, 1990.

[CM84] W. F. Clocksin and C. S. Mellish. *Programming in Prolog.* Springer-Verlag, 1984.

[CT87] Mark R. Cutkosky and Jay M. Tenenbaum. Cad/cam integration through concurrent process and product design. In *Winter Annual Meeting of the ASME: An Integrated Approach to Manufacturing Analysis and Sysnthesis*, Dec. 1987.

[DF89] L. De Floriani. Feature extraction from boundary models of three-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(8):785–798, Aug 1989.

[DW88] Xin Dong and Michael Wozny. Feature extraction for computer aided process planning. In *Proceedings of the Third International Conference on Computer-Aided Production Engineering, Ann Arbor, Michigan*, June 1988.

[FG89] Bianca Falcidieno and Franca Giannini. Automatic recognition and representation of shape-based features in a geometric modeling system. *Computer Vision, Graphics and Image Processing*, pages 93–123, 1989.

[Fle88] Alan Fleming. Geometric relationships between toleranced features. *Artificial Intelligence*, 37:403–412, 1988.

[Ful69] Walton Fulks. *Advanced Calculus, An Introduction to Analysis.* John Wiley and Sons, Inc., New York, 1969.

[Gra76] A. R. Grayer. The automatic production of machined components starting from a stored geometric design. In *Programming Languages for Machine Tools (PROLAMAT).* North Holland Publishing Company, 1976.

[Har69] F. Harary. *Graph theory.* Addison Wesley, Mass., 1969.

[Hay87] C. Hayes. Using goal interactions to guide planning. In *Proceedings of the AAAI-87; the Sixth National Conference on Artificial Intelligence*, pages 224–228, 1987.

[HB86] K. E. Hummel and S. L. Brooks. Symbolic representation of manufacturing features for an automated process planning system. Technical Report BDX–613–3580. Bendix Kansas City Division, 1986.

[Hen84] M. Henderson. *Extraction of Feature Information from Three Dimensional CAD Data.* PhD thesis, Purdue University, 1984.

[HM85] E. E. Hartquist and A. Marisa. *PADL–2 Users Manual.* Production Automation Project, University of Rochester, Rochester, New York, 1985.

[Hof89] Christoph M. Hoffmann. *Geometric and Solid Modeling, An Introduction.* Morgan Kaufmann Publishers, Inc., 1989.

[HQ82] Robert M. Haralick and David Queeny. Understanding engineering drawings. *Computer Graphics and Image Processing*, 20:244–258, 1982.

[Ide87] N. C. Ide. Integration of process planning and solid modeling through design by features. Master's thesis, University of Maryland, College Park, Department of Computer Science, 1987.

[Jak82] Ryszard Jakubowski. Syntactic characterization of machine parts shapes. *Cybernetics and Systems: An International Journal*, 13(1):1–24, 1982.

[JC88] S. Joshi and T. C. Chang. Graph-based heuristics for recognition of machined features from a 3d solid model. *Computer-Aided Design*, 20(2):58–66, Mar 1988.

[Jos87] S. Joshi. *CAD Interface for Automated Process Planning.* PhD thesis, Purdue University, 1987.

[Kar88] Michael Karasick. *On the Representation and Manipulation of Rigid Solids.* PhD thesis, Department of Computer Science, McGill University, Aug 1988.

[KJ87] T. Kramer and J. Jun. The design protocol, part editor, and geometry library on the vertical workstation of the automated manufacturing research facility at the national bureau of standards, 1987. Internal Report.

[KM76] K. Kuratowski and A. Mostowski. *Set Theory.* North Holland Publishing Company, 1976.

[Kum88] B. Kumar. *Feature Extraction and Validation within a Flexible Manufacturing Protocol.* PhD thesis, University of Maryland, College Park, 1988.

[Kun84] H. Kung. *An investigation into the development of process plans from solid geometric modeling representation.* PhD thesis, Oklahoma State University, 1984.

[Kyp80] L. K. Kyprianou. *Shape Classification in Computer-Aided Design.* PhD thesis, Christ's College, University of Cambridge, 1980.

[LDS86] S. C. Luby, J. R. Dixon, and M. K. Simmons. Design with features: Creating and using a feature data base for evaluation of manufacturability of castings. *Computers in Mechanical Engineering*, 5(3):25–33, 1986.

[Man88] M. Mantyla. *An Introduction to Solid Modeling.* Computer Science Press, College Park, MD, 1988.

[Men75] B. Mendelson. *Introduction to Topology.* Allyn and Bacon, Inc., 1975.

[Mil88] Victor Joseph Milenkovic. *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic.* PhD thesis, Department of Computer Science, Carnegie Mellon University, July 1988.

[Min85] R.H. Miner. A method for the representation and manipulation of geometric features in a solid model. Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, May 1985.

[Mor85] M. Mortenson. *Geometric Modeling*, chapter 9–13. John Wiley & Sons, 1985.

[Nau87] D. S. Nau. Automated process planning using hierarchical abstraction. *Texas Instruments Technical Journal*, pages 39–46, Winter 1987. Award Winner, Texas Instruments 1987 Call for papers on Industrial Automation.

[Nil80] Nils J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Company, Palo Alto, California, 1980.

[Pin82] C. Pinter. *A Book of Abstract Algebra.* McGraw–Hill Book Company, 1982.

[Pra87] M. J. Pratt. Form features and their applications in solid modelling. In *Tutorial paper on Advanced Topics in Solid Modelling at SIGGRAPH 1987*, July 1987.

[Pra88] M. J. Pratt. Synthesis of an optimal approach to form feature modelling. In *Proceedings of the ASME Computers in Engineering Conference*, Aug 1988.

[PS85] Franco P. Preparata and Ian Shamos, Michael. *Computational Geometry, An Introduction.* Springer-Verlag, New York, 1985.

[PSP89] J. M. Pinilla, Finger S., and F. B. Prinz. Shape feature description and recognition using an augmented topology graph grammar. In *Proceedings of the NSF Engineering Design Research Conference*, June 1989.

[RC86] A. A. Requicha and S. C. Chan. Representation of geometric features, tolerances, and attributes insolid modelers based on constructive geometry. *IEEE Journal of Robotics and Automation*, 2(3):156–166, September 1986.

[Req77] A. G. Requicha. Mathematical models of rigid solid objects. Technical Report TM-28, Production Automation Project, University of Rochester, Nov 1977.

[Req83] A. A. Requicha. Toward a theory of geometric tolerancing. *International Journal of Robotics Research*, 2(4):45-60, 1983.

[Rog89] Mary Rogers. Comparison of task1 functional requirements to task0 features technology state of the art. Technical Report R-89-GM-02, CAM-I Inc., 1989.

[ROM85] Shape Data Ltd. *The Romulus Solid Modeling System, V6.0 User's Reference Manual Version 1*, 1985.

[RT78] A. G. Requicha and R. Tilove. Mathematical foundations of constructive solid geometry : General topology of closed regular sets. Technical Report TM-27a, Production Automation Project, University of Rochester, June 1978.

[RV77] A. G. Requicha and H. B. Voelcker. Constructive solid geometry. Technical Report TM-25, Production Automation Project, University of Rochester, Nov 1977.

[RV85] A. G. Requicha and H. B. Voelcker. Boolean operations in solid modeling boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 73(1):30-44. 1985.

[SHA83] S. M. Staley, M. R. Henderson, and D. C. Anderson. Using syntactic pattern recognition to extract feature information from a solid modelling database. *Computers in Mechanical Engineering*, 2(2):61-65, 1983.

[Sim63] G. Simmons. *Introduction to Topology and Modern Analysis*. McGraw-Hill Book Company, 1963.

[SL87] R. Srinivasan and C. R. Liu. On some important geometric issues in generative process planning. In *Proceedings of the Winter Annual Meeting of the American Society of Mechanical Engineers, Boston, MA, December 1987*, pages 229-244, 1987.

[Smi89] T. Smithers. Ai-based design versus geometry-based design or why design cannot be supported by geometry alone. *Computer-Aided Design Journal*, 21(8):141-149, April 1989.

[SR88] J. J. Shah and M.T. Rogers. Functional requirements and conceptual design of the feature-based modeling system. *Computer-Aided Engineering Journal*. 7(2):9-15, Feb 1988.

[SRSM89] Jami Shah, Mary Rogers, Palat Sreevalsan, and Abraham Mathew. Functional requirements for feature based modeling systems. Technical Report R-89-GM-01, CAM-I Inc., 1989.

[SS86] Leon Sterling and Ehud Shapiro. *The Art of Prolog*. The MIT Press, 1986.

[Ste84] G. L. Steele Jr. *Common Lisp: The Language*. Digital Press, Burlington, MA. 1984.

[Str80] I. A. Stroud. The build picture book. Technical Report CAD Group Document 104, Computer Laboratory, University of Cambridge, England. 1980.

[SW86] B. Smith and J. Wellington. Initial graphics exchange specifications (iges) version 3.0. Technical Report NBSIR 86-3359, National Bureau of Standards. Gaithersburg MD. 1986.

[TC89] Jay M. Tenenbaum and Mark R. Cutkosky. First-cut: A computational framework for rapid prototyping and team design. In *Proceedings of the AAAI Spring Symposium on AI in Manufacturing*, March 1989.

[Tex87a] Texas Instruments Incorporated. *Explorer Lisp Reference Manual*, June 1987.

[Tex87b] Texas Instruments Incorporated. *Explorer TI Prolog User's Guide*, Oct. 1987.

[Tho89] Scott Thompson. Environment for hierarchical abstraction: A user guide, May 1989. Master's Scholarly paper, University of Maryland, Department of Computer Science.

[Tur88] Joshua U. Turner. Automated tolerance using solid modeling technology. In *Proceedings of AUTOFACT'88, Chicago*, 1988.

[Van89a] G. Vanecek Jr. Protosolid: An inside look. Technical Report CSD-TR-921, Purdue University, Computer Sciences Department, October 1989.

[Van89b] G. Vanecek Jr. *Set Operations on Volumes Using Decomposition Methods.* PhD thesis, University of Maryland, College Park, 1989.

[Van90] Jan Vandenbrande. *Automatic Recognition of Machinable Features in Solid Models.* PhD thesis, University of Rochester, Electrical Engineering Department, 1990.

[VDZS85] M. Vaghul, J. R. Dixon, G. E. Zinmeister, and M. K. Simmons. Expert systems in a cad environment : Injection molding part design as an example. In *Proceedings of the 1985 ASME Conference on Computers in Engineering*, 1985.

[Wei86] Kevin Weiler. *Toplogical Structures for Geometric Modeling.* PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, 1986.

[Win84] Patrick Henry Winston. *Artificial Intelligence.* Addison-Wesley Publishing Company, 1984.

[Woo82] T C. Woo. Feature extraction by volume decomposition. In *Proceedings of the Conference on CAD/CAM technology in Mechanical Engineering*, 1982.

[ZH90a] G. M. Zhang and T. W. Hwang. Analysis and control of geometric tolerancing through surface topography generation. In *Symposium on Automation of Manufacturing Processes. 1990 ASME Annual Meeting. Dallas, Texas*, 1990.

[ZH90b] G. M. Zhang and T. W. Hwang. Analysis of the cutting dynamics in microscale. In *Symposium on Fundamental Issues in Machining. 1990 ASME Winter Annual Meeting, Dallas, Texas*, 1990.