NASA Technical Memorandum 86241

A Rule Based Computer Aided Design System

Timothy Premack

OCTOBER 1986

NASA

NASA Technical Memorandum 86241

A Rule Based Computer Aided Design System

Timothy Premack
Goddard Space Flight Center
Greenbelt, Maryland

National Aeronautics
and Space Administration

Scientific and Technical Information Branch

1986

TABLE OF CONTENTS

PREFACE

This paper was originally submitted to the John Hopkins University as a master's essay in October, 1985.

I would like to thank Drs. A. Douglas and J. O'Rourke of Johns Hopkins University for advising me on this thesis. I also thank L. Purves of The Goddard Space Flight Center for providing an environment to conduct this research, and Prologica for supplying a bug free Prolog interpreter.

A RULE BASED COMPUTER AIDED DESIGN SYSTEM

Timothy Premack

INTRODUCTION

A theory of the process of design and how a design is represented is presented. A description of current CAD systems and their shortcomings is described. A system is then presented which addresses the needs of design: it models the iterative nature of the design process, the associativity between constitutive parts, and permits design by an explicit function of parameters. An example design of a spring piston is used to illustrate the system.

MODELING DESIGN

To model design one must not only represent the physical design but model the design process itself. A model of the design is a description of the geometry representing its constitutive parts and their relationship to each other in their assembled configuration.

The process of design is usually defined as iterative [1]. It consists of specification, initial design, evaluation of the design, and redesign. In essence, given a requirement, one produces a design and refines it until it meets specific requirements.

The relationship of the parts forming an assembly resembles a hierarchy [2]. The top node of the tree is the assembly itself, the branches form sub-assemblies, moving down the tree until the assembly has been subdivided down into indivisible parts, i.e. one piece of geometry. If the Constructive Solid Geometry* (CSG) method is used to describe the geometry, then the hierarchy can move down even further. For example one might think of a bolt as being a quantum part, but it can be modeled as the union of a cylinder and a hexagonal prism.

A physical object can be described as an interconnecting relationship between many ideas [3]. In applying this theory to design, one can see how various parts are linked together. Where two parts interact, each affect the design of the other. A simple example is a bolt in a hole. The information shared between the two parts is the diameter of the hole or the diameter of the bolt. Therefore an assembly of parts can be represented as a directed graph.

*Constructive Solid Geometry is a geometric modeling technique which generates complex geometric objects for the logical combination (union, negation, intersection) of solid geometric primitives (sphere, cube, cone, etc.).

CAD SYSTEM AND THEIR LIMITATIONS

Current commercial CAD systems are geometric data base editors. The geometries are interactively entered into the system via commands from a keyboard, graphics tablet, or mouse. There are also commands to manipulate and display the data base. The programs use the geometric data along with other information to perform analysis such as interference checking, mass properties, mechanism simulation, and structural analysis.

CAD systems do not have methods for storing the associativity of elements in their data base. Many systems have methods for grouping parts into one part. This makes the data base smaller and manipulation of the parts easier for the user. For dimensioning parts, systems can tie the dimension lines to the part geometries, so if the geometry is changed, the annotation changes automatically. The journaling capability can also be considered to provide associativity between parts. A journal file is made from the user's interactive commands. If there is a change to be made, the user can edit the journal file and then execute it. Other systems provide what are called parameterized parts, a macro capability for defining solid geometries in the same format as the system uses.

These methods provide the support needed for design, but they are deficient in two key areas. They do not explicitly support the iterative design process nor store limited part associativity. An example of their limitations follows.

A truss structure is defined as four elements joined to represent a rectangle with a fifth element as a diagonal (an element being some prismatic bar). If the length of one of the sides is changed, the element joining the sides does not change position or length, and the connectivity between elements in the data base is not stored.

To solve this problem, one could define the truss to be a parameterized part. This definition has limited usefulness, however. The truss does not change with a change in definition, but must be removed and then recreated. Hence the idea of parameterized parts is useful mainly in data base creation, not in modification.

One could also use a journal file to define the truss. The designer edits the journal file to define the new truss, removes the current truss from the data base and execute the journal file—a process similar to the parameterized part. Although the journal file protects the user from loss of work, its use as a design generator has some problems. For one, there is a large overhead in interpreting the file to create a design. If used as a design generator the file actually becomes the data base, from which the system creates its internal data base. Since the journal file is a sequentially ordered list of commands, the user is responsible for inserting commands in the correct order. For menu driven systems this procedure is quite complicated, since the function of a command is predicated on the previous command.

In designing the next iteration one favors the use of as much of the data base as possible in developing the next design. In CAD, as in board design, the process is a manual one. The designer must manually change the drawing to reflect new design considerations. Since the systems store limited part associativity, the designer must manually change all the affected parts in the event of a design change. Needed is a system that stores the part connectivity and uses as much of the design data base as possible between iterations.

In summary, the problem with current CAD systems is they retain virtually none of the logic that controls the geometric shapes in the database, they are design documentors [4]. They provide no more design support than current text editors provide composition support. The analysis programs are equivalent to spelling checkers.

SYSTEM REQUIREMENTS

The functions a CAD system should exhibit in providing additional design support are now described.

DESIGN AS A FUNCTION

A mechanical design can be thought of as a set of mathematical functions where each function describes a dimension in the design. Some of these functions are constants, these can be thought of as design parameters. Therefore given n design functions $f_1, \ldots, f_n$, k design parameters $p_1, \ldots, p_k$, one can define m dimensional parameters $d_1, \ldots, d_m$ [5]. Depending on the design requirements each dimensional parameter d_i is dependent on a subset of the design parameters p and other dimensional parameters d or:

$$d_i = f_i \{ d_1, \ldots, d_m, p_1, \ldots, p_j \}$$

NETWORK DATA BASE FOR CAD

This type of description can be represented as a directed graph, where each node is d_i, the i^{th} dimensional parameter or p_k the k^{th} design parameter, and the paths from the other nodes are the arguments to the function f_i. The idea or concept of the design is represented by the network. An execution of the design is produced by assigning numerical values to the parameters.

Note that each path is directed. For example a bearing in a pillow block, either the bearing defines the pillow block or vice versa. They cannot mutually define one another. One can see from the network that any change in a design parameter ripples through the design, generating a new execution of the idea represented by the network [6].

An example of a CAD system using a network data base and nodal values to define the solid geometries representing each part would be:

$$d_1 = f_1 \{ p_1, p_2, p_9, \ldots \}$$

$$d_2 = f_2 \{ p_1, d_1, \ldots \}$$

$$d_3 = f_3 \{ p_2, d_1, p_4, \ldots \}$$

.

.

.

part1 is a cylinder with radius d_1, length d_2, centered at (d_3, d_4, d_5).

Of course d_3 can not be used in a function until it has been defined. The functions can consist of anything, strength of material laws, kinematics, dimension matching, random number generation, etc.

CONTROL STRUCTURE

If one represents the design as a network, what type of control structure would a program require? What types of ancillary functions would be needed to support the design? In describing the dimensional parameters, a procedural math based language is needed, something like FORTRAN, BASIC or APL would suffice.

Most designs require the choice of some standard parts, i.e. structural steel, bolts, bearings etc. Since the method most useful for storing tabular data is a relational data base, this ingredient is included with the procedural language.

But what about checking for legal dimensional parameters? Conditionals must be provided. That is not a problem—each function can check for legal values. What happens if a dimensional parameters is not legal? Does the design fail? If a design fails in trying to satisfy dimensional parameters, the control mechanism should back up and retry the previous function since it may have more than one solution available. What happens when more than one standard part will work in the design, implying a multiplicity of executions? The system should then generate all possible represented designs.

LOGIC PROGRAMMING

A logic programming language called Prolog provides the three functions required to represent the design as a network: math based procedural language, relational data base, and automatic generation of all possible solutions. Prolog uses first order predicate calculus to represent its data and resolution theorem proving to prove data base queries. The resolution theorem prover performs a depth first search in order to resolve the predicates. For a description of the algorithm see [7]. For a description of the Prolog language see [8].

Although Prolog is not considered a mathematical language it contains all the FORTRAN intrinsic functions such as trigonometry, logarithms, bit manipulation, etc. Although APL is probably the best interactive language for vector or matrix operations, Prolog can work with lists of numbers. Inherent in Prolog is a relational data base. By defining predicates with the same name, a table can be created and searched. The generation of all possible solutions is also built into Prolog. The inference algorithm will generate all possible solutions to a problem represented in the data base.

Prolog Examples—Prolog predicates are broken down into two types, facts and rules. A fact is an assertion about something, such as the sky is blue. In Prolog this would be represented as:

Example of a Prolog Fact.

```
sky(blue).
```

A rule is an assertion about something, such as the factorial of the number N is N times the factorial of N-1. In Prolog this would be:

Example of a Prolog Rule.

```
factorial(N,Answer) :-
  N1 is N-1,
  factorial(N1,ANS1),
  Answer is ANS1 * N.
```

This is a recursive definition of factorial. It needs a fact as a terminal case, that is the factorial of one is one.

Prolog Definition of Factorial.

```
factorial(1,1).
factorial(N,Answer) :-
  N1 is N-1,
  factorial(N1,ANS1),
  Answer is ANS1 * N.
```

LOGIC PROGRAMMING IN DESIGN

Logic has been shown to be a good method for performing design [13], and is being used in other systems. The use of a rule data base for CAD has been demonstrated in various applications consisting of architectural design, crankcase design, and v-belt drive design.

The Carter system [9] uses a rule data base to design crankcases. Although this is not a general design program and does not provide graphical output, it does illustrate the use of a rule data base for design. The v-belt drive design program also uses a rule data base. It uses OPS5 instead of Prolog. In architectural design, Swinson describes the use of a fact dependency system, [11] and [12]. Another example of an architectural application isdemonstrated by Markusz, [13]. It uses Prolog to define a data base for the design of apartments.

Although the systems describe are not general design programs, they all use the idea of describing the design with logic.

A "SATISFICING" SYSTEM

Simon coined the word "satisficing" [14] to describe "decision methods that look for good or satisfactory solutions instead of optimal ones". The CAD system proposed is a satisficing system. Given the design rules it will generate all the possible designs represented by the rules. This is different from the idea of optimization. Optimization finds the values of the design variables which will yield the optimal value of an objective function. The classic example of an objective function in structural optimization is weight minimization.

RULE BASED CAD SYSTEM

The CAD system uses an entirely different data base than conventional systems. To represent the geometry, the data base consists of three parts:

1. Predicates describing each node of the network. These are the design functions.
2. Predicates which define the geometry in a parameterized form.
3. Predicates which describe the hierarchical structure of the assembly.

To display a design, the design predicate would have to be proven and the parameters passed to the geometric definitions for display.

DATA BASE

The system uses Prolog's predicates as the data base. It is created directly by the user. The predicate's name defines its data type. The data types supported are:

- DESIGN: A design
- PART: Hierarchical Structure.
- Geometric Data Types
  —REV_CLOSED: Solid of Revolution.
  —REV_OPEN: Solid of Revolution.
  —SLE: Solid of Linear Extrusion.
  —SHE: Solid of Helical Extrusion.

The arguments to the predicate consist of the following:

- Name: Any legal Prolog structure* or atom the user wishes to identify the data element. Examples are "hello", "123", "0.5", "part(34)".
- Attribute List: A Prolog list of structures containing auxiliary data. Such data items consist of color, density, reflectivity, or user documentation. It is open to the user to put any type of data the user deems useful.
- Parameter List: This argument is used by the system to pass the parameters to the data section. If the element does not have any parameters it should be instantiated (defined) as a Prolog empty list, i.e. [], else it is a variable.
- Data: The data structure contains various structures depending upon the element's data type defined by the predicate's name. The system uses the data stored here to display what ever is defined.

DESIGN FUNCTION

The design function is executed in a design predicate. The data to this predicate has two arguments. One is a predicate which returns a list of parameters. The second is a list of element names (corresponding to the name field) used in the design. To generate all the designs represented by the rules, the program uses Prolog's built in backtracking. It calls the design function to get the parameter list, displays the elements, and then by intentionally failing incurs the backtracking algorithm of the resolution theorem prover to try to prove the predicate a different way.

```
design( Name, Attributes ,
        data(
        Design_predicate, elements( element_1, ...,
        element_n )
          )).
```

Geometry—The system defines solids by sweeping a planer closed curve down another curve. This is one of many methods for representing solid geometry for CAD [16]. It was chosen for its compactness.

All the geometric definitions have two data items in common. They all have a planer curve definition and all are rotated and translated by a general transformation matrix. See appendix A for examples of each definition.

*A Prolog structure is essentially a named list, e.g. radius (1.5, inches) is a structure of arity two. Its name or principal functor is "radius", a Prolog atom. Structures may contain other structures.

Solid Of Revolution—

```
rev_open( Name , Attributes , Parameters,
        data( angle( Theta ),
                rf( Offset(X,Y,Z), angles( Yaw, Pitch,
                Roll)),
                curve( curve_definition )
          )).

rev_closed( Name , Attributes , Parameters,
        data( angle( Theta ),
                rf( Offset(X,Y,Z), angles( Yaw, Pitch,
                Roll)),
                curve( curve_definition )
          )).
```

Solid Of Linear Extrusion—

```
sle( Name, Attributes, Parameters,
        data( delta( Delta_X, Delta_Y, Delta_Z),
                rf( Offset(X,Y,Z), angles( Yaw, Pitch,
                Roll)),
                curve( curve_definition )
          )).
```

Solid Of Helical Extrusion—

```
she( Name, Attributes, Parameters,
        data(
                length( L ), mean_radius( R ), pitch( P ),
                rf( Offset(X,Y,Z), angles( Yaw, Pitch,
                Roll)),
                curve( curve_definition )
          )).
```

Part. A part is a hierarchical structure. It is an element which contains other elements. Along with passing the parameters to the elements in the list it can also define local parameters to the elements. Local parameters are stored in the element_parameters list structure. It can be used to define standard parts.

```
part( Name, Attributes, Parameters,
        data(
                rf( Offset(X,Y,Z), angles( Yaw, Pitch,
                Roll)),
                elements( element_1, element_2 ,
                ... , element_n) , Element_Parameters
          )).
```

Structure Definitions—

- rf(Offset(X,Y,Z), angles(Yaw, Pitch, Roll)) is a general transformation matrix. The part is rotated by the angles Yaw, Pitch, Roll, (Theta-z, Theta-y, Theta-x) and then is translated by X,Y,Z. The angles are in degrees. The XYZ convention is used to define the rotation matrix [17].
- curve_definition is a closed curve defined in the x-y plane. It consists of a list of verticies and circular arc definitions. The starting and ending verticies must coincide for a closed curve arc and it must be defined in the counter-clockwise direction.
- A vertex is described as v(X,Y), a point in the x-y plane.
- A circular arc drawn counter-clockwise is described as arc(s(SX,SY),c(CX,CY),e(EX,EY)). Where (SX,SY) is the start point of the arc, (CX,CY) is the center of the arc, and (EX,EY) is the end point of the arc. All the points lie in the x-y plane. A clockwise arc is described using the structure name arc_cw instead of arc.

EXECUTION

The CAD system runs on a VAX 11/780 under VMS version 3.4 operating system. It uses a Prolog interpreter marketed by Prologica located in King of Prussia, Pennsylvania. For a description of the Prolog interpreter see [18]. The interpreter is virtual memory based and can interface with other languages. This feature was used in the system to provide the display interface to the various graphical devices.

The Prolog part of the system defines the data base and generates the model for display. A FORTRAN sub-system is used in mapping the three dimensional world model to the two dimensional screen. The Prolog consists of approximately 1300 lines of code and documentation, the FORTRAN part consists of approximately 3000 lines of code and documentation. See appendix C for a list of the main body of the Prolog code which produces a design from the user defined predicates.

PISTON EXAMPLE

A spring loaded piston is used to demonstrate the CAD system. It is a piston in a cylinder with two opposing springs. The device was used in a robotics experiment and was designed on a commercial CAD system in 1981. See appendix B for the Prolog predicates which define the piston. The model produced by the rule based CAD system is shown in figures 7,8 and 9 with the element ''housing'' shown sectioned for clarity. Only one of the four possible designs is shown. Table 1 contains the parametric values defined by the design predicate 'piston'. The names shown in this table correspond to the structure definitions defined in the predicate 'piston'. These structure definitions also correspond to the structure definitions in the solid geometric definitions.

DESCRIPTION OF THE PISTON DESIGN

The design is parameterized by five parameters:

- Minimum travel.
- Load range, minimum and maximum.
- Rod diameter.
- Wall thickness.

The choice of design parameters was not arbitrary, but the set shown is not necessarily the fundamental set. The wall thickness is dependent on the load range. Along with design parameters there are certain functional and geometric constraints which must be met. These were formulated in a predicate called ''constraint''.

Design Constraints

1. Spring must meet load constraints.
2. Rod must fit inside the spring.
3. Spring must have desired travel.
4. Bearings must fit the rod.

Note for item 4 how Prolog is used to search for an explicit match. The predicate ''ball_bushing'' also instantiates the dimensions of the ball bushing.

In the section defined as ''spring data'' note that more than one spring was defined. Stock springs were used to demonstrate the relational data base capability of Prolog and how it was used in the CAD system.

The predicate ''piston'' is the user defined design predicate which defines all the dimensions required for the geometric elements. It first instantiates the design parameters by calling the ''spec''predicates. The rest of the predicate defines the other geometric parameters and checks if they are valid by use of the ''constraint'' predicate. Inter-part connectivity is accomplished by arithmetic expressions defining parameters to be passed to the geometric definitions. Some examples of connectivity in the piston design are:

- Inside diameter of piston housing defined by the outside diameter of the spring.
- The length of the piston housing is defined by the springs, desired travel, bearing length, and housing thickness.
- The rod length is defined by the bearing spacing, bearing lengths, and compressed lengths of the springs.

The directed graph representing the design connectivity is shown in figure 6. The variable names correspond to the variable names used in the predicate ''piston'' (variables begin with a capital letter).

PISTON REDESIGN

New design possibilities are performed by changing the ''spec'' predicates or by changing the design predicate or by adding more standard parts to the data base such as springs and bearings. If the predicate ''spec(travel(1.0))'' is changed to ''spec(travel(1.5))'' the only design possibility is design possibility 2 in table 1.

To illustrate how a modification was performed on the design, two parts, internal_cap and lvdt_cap, were modified. Instead of butting against the rod a spot face was added to the caps so the rod fits in a blind hole. Appendix B contains the code for the complete definition of the redefined piston with new or modified predicates italicized. Table 2 contains the new parameters for the design and figures 10, 11 and 12 show the new piston. The only solid geometry definition changed was that of the element called ''cap''. Three equations were changed to reflect the addition of the spotface and one predicate was added to relate the spotface depth with the rod diameter.

The length of the Prolog code in appendix B may seem unduly long to represent the eight parts making up the piston. More than just geometries are defined, the logical connection between parts is represented, a relational data base of stock springs is defined, and the geometry is represented in parametric form.

CONCLUSION

The rule based CAD system described provides three functions of a design system by: 1. supporting the iterative nature of the design process, 2. defining the associativity between parts in a design, and 3. allowing explicit design by parameters and functions. As can be seen from the piston example, new design executions can be obtained by changing the "spec" predicates. Iterations are performed by refining the rule data base and the geometric elements are described explicitly as a function of design parameters.

Prolog was used because it is a rule based language. Using its resolution theorem prover to resolve the design represented in its rule data base, it automatically generates all possible solutions. Its predicates and list structures provide a flexible representation of the solid models and of the design functions.

In preliminary design, much of the design time is spent examining the effect parameter changes have on the design's configuration. If the connectivity between mating parts is defined in the data base, a new design can be generated easily by changing a parameter or a rule. This increases the efficiency of the process.

The goal of this research is to develop a technique for design synthesis. In design synthesis, the concept is described in physical functions and capabilities rather than in geometric terms. The CAD system developed here is a step toward that goal in that it was used to develop the kinds of data structures and knowledge representations that define the required geometric interrelationships of a mechanism.

The next steps in this research requires use of this program as a design generator, the incorporation of test functions to evaluate the designs and the heuristics to alter the rule data base. Design synthesis could then be realized through the use of generic design rules, a data base of parts and a translator to convert functions and capabilities to geometric terms.

REFERENCES

1. Dixon, J.R., M.K. Simmons, P.R. Cohen, An Architecture for Application of Artifical Intelligence to Design, *ACM IEEE 21st Design Automation Conference Proceeding 1984.*
2. Katajma, Katsuhiro, Hiroyuki Yoshikawa, HIMADES-1: A Hierarchical Machine Design System Based on the Structure Model for a Machine, *Computer Aided Design*, Volume 16, number 6, 1984.
3. Alexander, Christopher, *Notes on the Synthesis of Form*, Harvard University Press, Cambridge, Massachusetts, 1970.
4. Sambura, Andrzej S., and John S. Gero, Framework for a Computer Integrated Design Environment, *Proceeding of the IFIP WG 5.2 Working Conference on CAD Systems Framework*, Roros, Norway, 15-17 June 1982, Edited by K. BO and F.M. Lillehagen, North-Holland 1983.
5. Asimow, Morris, *Introduction to Design*, Prentice-Hall, 1962.
6. Hiroyuki Yoshikawa, CAD Framework Guided by General Design Theory, *CAD Systems Framework*, Proceeding of the IFIP WG 5.2 Working Conference on CAD Systems Framework, Roros, Norway, 15-17 June 1982, Edited by K. BO and F.M. Lillehagen, North-Holland 1983.
7. Nilsson, Nils J., *Principles of Artificial Intelligence*, Tioga Publishing Co., 1980.
8. Clocksin, W.F., Mellish, C.S., *Programming in Prolog*, Springer-Verlag, 1981.
9. Reynier, Marie, Fouet, Jean-Marc, Automated Design of Crankcases: The Carter System, *Computer Aided Design*, Volume 16, Number 6, November 1984.
10. Dixon, J.R., Simmons, M.K., An Architecture for Application of Artificial Intelligence to Design, *ACM IEE 21st Design Automation Conference*, Proceedings '84.
11. Swinson, Peter S.G., Fernando C.N., Bijl, Aart, A Fact Dependency System for the Logic Programmer, *Computer Aided Design*, Volume 15, Number 6, November 1983.
12. Swinson, Peter S.G., Prolog: A Prelude to a New Generation of CADD, *Computer Aided Design*, Volume 15, Number 6, November 1983.
13. Markusz, Asuzsanna, Design in Logic, *Computer Aided Design*, Volume 14, Number 6, November 1982.
14. Simon, Herbert A., *The Sciences of the Artificial*, MIT Press, 2nd Edition, pp 138-139.
15. Schmit, Lucien A., Structural Synthesis - Its Genesis and Development, *AIAA Journal*, Volume 19, Number 10, October 1981.
16. Aristides A.G. Requicha, Representations for Rigid Solids: Theory, Methods, and Systems, *Computing Surveys*, Volume 12, Number 4, December 1980.
17. Goldstein, Herbert, *Classical Mechanics*, Addison-Wesley, Second Edition, July 1981.
18. *Prolog-1 Language Reference Manual*, Issue 2, March 1984, Expert Systems Limited, 9 West Way, Oxford OX2 OJB

APPENDIX A

ELEMENT DEFINITIONS

This appendix contains examples of element definitions along with the solid geometry they represent.

```
sle( figure1 ,
        [ comment(' Example of user documentation') ] , [] ,
        data(
                delta( 0.1E1 , 1 , 5.5),
                rf( offset( 0,0,0), angles(0,0,0)),
                curve( v(0,0), arc( s(4,0), c(4,2), e(4,4)),
                        v(0,4), v(0,0)))).
```

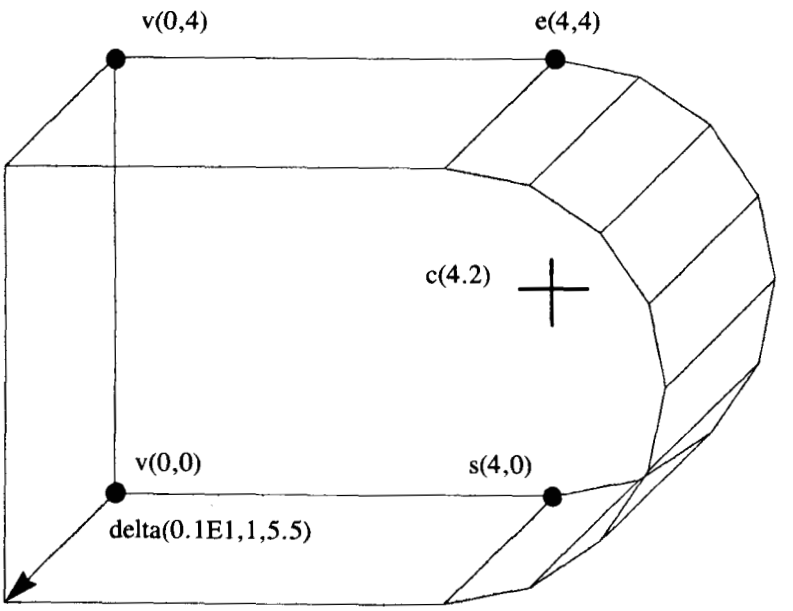


Figure 1. Solid of Linear Extrusion.

```
rev_closed( figure2, [], [] ,
        data(
                angle(36.0),
                rf( offset( 0,0,0), angles(30,45,0)),
                curve(
                        v(0,Y), v(4,Y),
                        arc_cw( s(4,YP) , c(2,YP), e(0,YP) ),
                        v(0,Y)
        ))) :- Y is 3.0, YP is Y + 4.0, !.
```

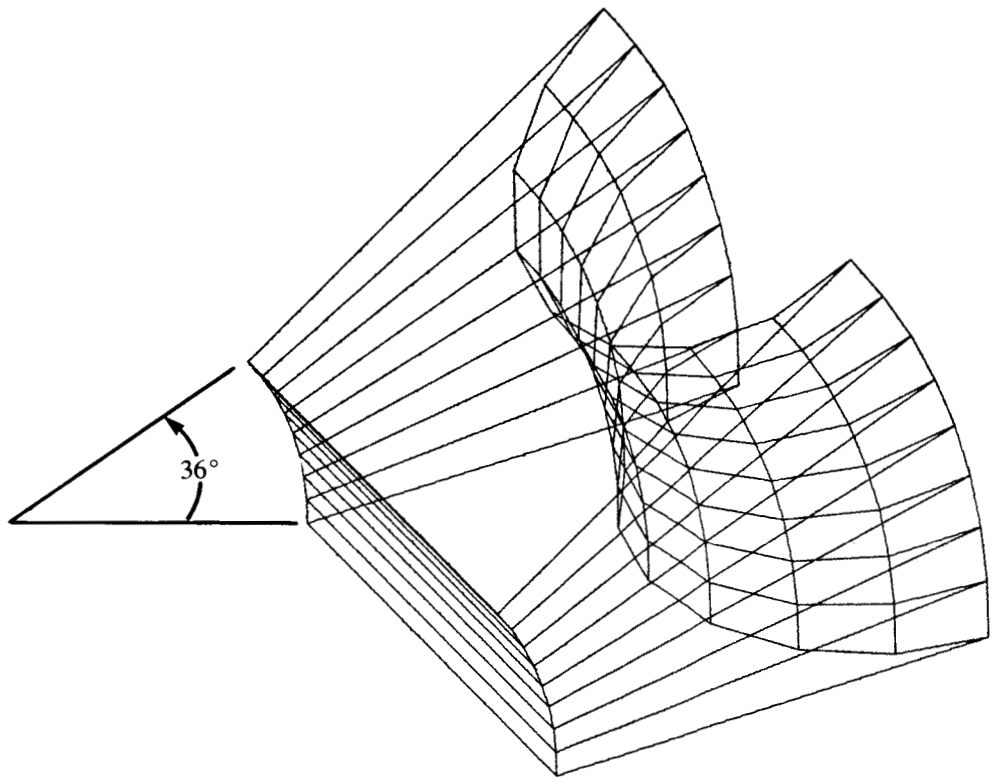


Figure 2. Solid of Revolution - Closed Curve.

```
rev__open( figure3 , [] , [] ,
        data(
                angle(360.0),
                rf( offset(0,0,0), angles(30,45,0)),
                        curve( arc( s(5,0), c(2.5,0), e(0,0))
                ))).
```

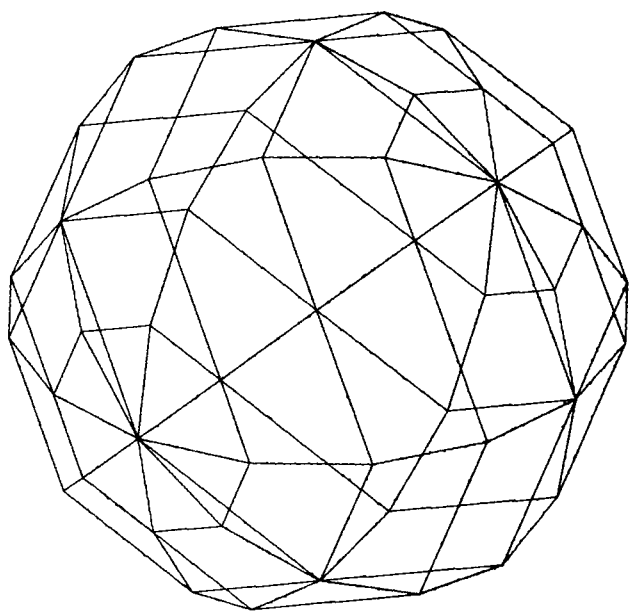


Figure 3. Solid of Revolution - Open Curve.

```
part( figure4 , [] , [] ,
        data(
                rf( offset(1,2,3), angles( 0,0,0)),
                elements( ball, tube ),
                [ ball(center([0,0,0]), radius(C)),
                  tube( id(ID), od(OD), length(L)) ]
        )) :-
                C is 2.0,
                ID is C * 2 , OD is ID + 1, L is C*3, !.
rev_open( ball, [] , Parameters , data(
                angle(360.0),
                rf( OFFSET , angles(0,0,0)),
                curve(
                        arc( s(RAD,0), c(0,0), e(-(RAD),0))
           ))) :-
                param( ball(center(CENT), radius(RAD)),
Parameters),
                OFFSET =.. [offset|CENT], !.
rev_closed( tube , [] , Paramters , data(
        angle(360.0),
        rf( offset(0,0,0), angles(0,0,0)),
        curve( v(X1,Y1), v(X2,Y1), v(X2,Y2), v(X1,Y2), v(X1,Y1)
))) :-
        param( tube( id(ID), od(OD), length(L)), Paramters ),
        X1 is 0.0,
        Y1 is ID / 2,
        X2 is L + X1,
        Y2 is OD * 0.5, !.
```

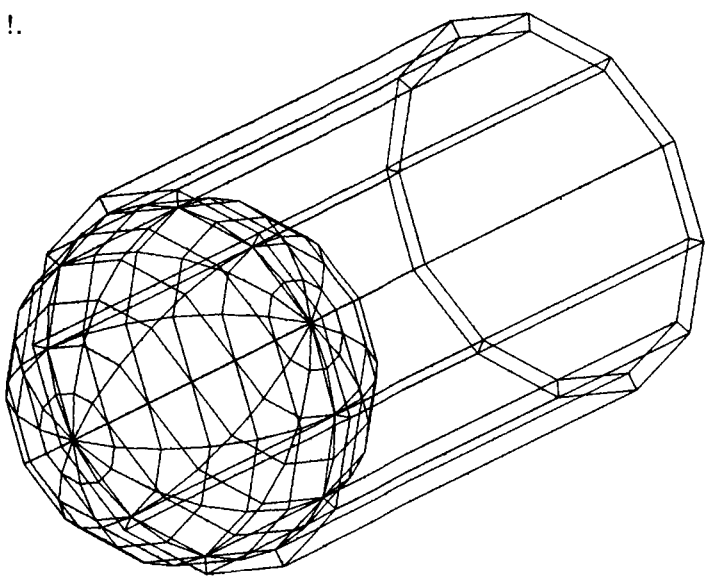


Figure 4. Part Definition with Parameters.

```
part( figure5, [] , [] ,
        data(
        rf( offset(0,0,0), angles(30,75,0)),
        elements( spring), [ spring( od(4), wire_dia( 0.25 ),
        pitch(1), length(4)) ] )) :- !.
she( spring, [], Params,
        data(
        length(L), mean_radius(R), pitch(P),
        rf( offset(0,0,0), angles(0,0,0)),
        curve( arc( s(SX,0), c(0,0), e(SX,0))
                )
        )) :-
        param( spring( od(OD), wire_dia(WD), pitch(P),
                length(L)), Params),
        R is (OD -WD) * 0.5,
        SX is WD / 2.0, !.
```

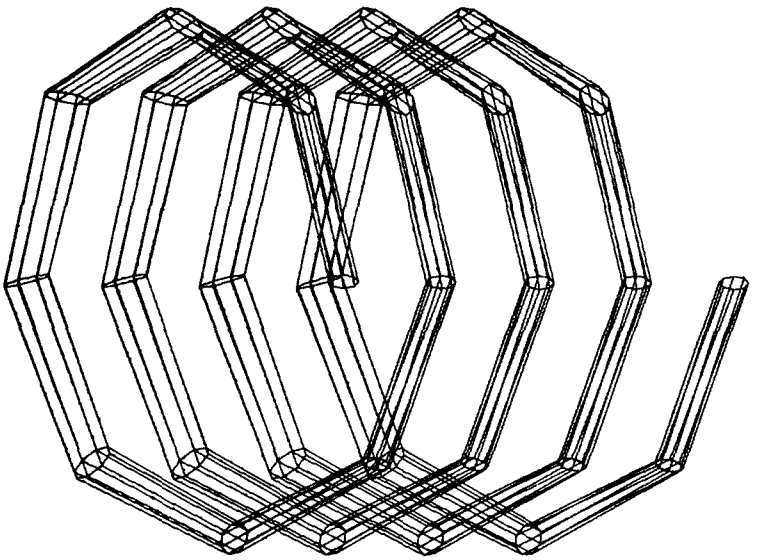


Figure 5. Solid of Helical Extrusion.

APPENDIX B

SPRING PISTON EXAMPLE

```
new :- reconsult(piston1), consult(userlib), !.

/*
                    DESIGN SPECIFICATIONS
*/

                /*      At least this Travel      */
spec(travel(1.0)).
spec(load_max(30.0)).                                   /*      At solid height */
spec(load_min(10.0)).
spec(rf(offset(0.0,0.0,0.0),angles(0.0,0.0,0.0))).
spec(thickness(0.25)).
spec(identity(
        rf(offset(0.0,0.0,0.0),angles(0.0,0.0,0.0))
        )).
spec( rod,diameter(0.375)).
spec( rod,offset(0.0)).

/*
                    DESIGN CONSTRAINTS
*/

constraint( load , Load_at_solid_height, Max_load, Min_load ) :-
        Load_at_solid_height < Max_load,
        Load_at_solid_height > Min_load.

constraint( travel, Travel , Free_length , Solid_height ) :-
        Travel < Free_length - Solid_height.

constraint( rod_and_spring , Spring_ID, Rod_diameter ) :-
        Spring_ID > Rod_diameter.


design( piston , data(
        piston , elements(rod,bearing1,bearing2,
        internal_cap,lvdt_cap ,housing, spring1, spring2) )) :-
         !.

/*
                    PISTON DESIGN PREDICATE
*/
```

```
piston( [
        rod( length(Rod__length)),
        rod( diameter( Rod__diameter)),
        rod( offset( Rod__offset)),
        spring1( offset( S__X)),
        spring( id( Spring__ID)),
        spring( od( Spring__OD)),
        spring( length( Compressed__length)),
        spring2( offset( S__X2)),
        bearing( id( Bearing__ID)),
        bearing( od( Bearing__OD)),
        bearing( len( Bearing__len)),
        bearing1( offset( B__ox1)),
        bearing2( offset( B__ox2)),
        housing(
                radii(
                        H__spring__radius, H__bearing__radius,
                        H__rodclear__radius, H__outside__radius),
                lens( H__springtravel , H__bearings, H__rod ),
                rf( offset(H__offset,0,0) , angles(0,0,0))
                ),
        cap( thickness( Cap__thickness), od( Cap__OD) ),
        cap__rf( offset( 0,0,0), angles(0,0,0)),
        lvdt__cap__rf( offset(LVDT__offset,0,0) , angles(0,180,0) )

        ] ) :-

        spec( load__max( Load__maximum)),
        spec( load__min( Load__minimum)),
        spec( travel( Travel)),
        spec( thickness( Thick)),

        spec( rod,diameter(Rod__diameter)),
        spec( rod,offset(Rod__offset)),

        spring(Spring__name, od(Spring__OD),hd(Spring__hd),
                wd(Spring__wiredia),load__sh(Load__at__solid__height),
                fl(Free__length), k(Spring__K), sh(Solid__height) ),

/*      Find a valid spring         */

        constraint( load, Load__at__solid__height , Load__maximum ,
                                        Load__minimum ),

        constraint( travel, Travel , Free__length , Solid__height ),

        ball__bushing(Type ,bore(Rod__diameter),od(Bearing__OD),
                        len(Bearing__len)),

        Spring__ID is Spring__OD - 2 * Spring__wiredia,

        /* Rod must fit inside the spring */
        constraint( rod__and__spring , Spring__ID, Rod__diameter ),
```

```
/*
```

DEFINE PART ASSOCIATIVITY

```
*/

        Compressed__length is Free__length * 0.45 ,

        Distance__between__bearings is 3 * Bearing__len,
        S__X is Rod__offset ,

        S__X2 is S__X + Compressed__length +
                Distance__between__bearings + Thick,

        Rod__length is
                Distance__between__bearings
                + (Compressed__length * 2.0) + Thick ,

        Cap__thickness is Thick,
        Cap__OD is Spring__hd,

        Bearing__ID is Rod__diameter,
        B__ox1 is Compressed__length + S__X,
        B__ox2 is B__ox1 + 2.0 * Bearing__len,

        H__spring__radius is ( Cap__OD * 0.5 ) + 0.1,
        H__bearing__radius is Bearing__OD * 0.5,
        H__rodclear__radius is Rod__diameter * 0.5 + 0.010,

        H__springtravel is Cap__thickness + Compressed__length
                                                + Travel,

        H__bearings is Distance__between__bearings ,
        H__rod is Thick,
        H__outside__radius is H__spring__radius + Thick,
        H__offset is -(H__springtravel) + Compressed__length ,
        LVDT__offset is Rod__length.

/*
```

LINEAR BEARING DATA BASE
THOMPSON LINEAR BEARING CO

```
*/

ball__bushing('XA-61014',bore(0.375),od(0.6250),len(0.875)).

/*
```

STOCK SPRING DATA BASE
LEE SPRING COMPANY

```
*/

spring('LC-055J-6', od(0.720),hd(0.75),wd(0.055),
                load__sh(13.0),fl(1.75),k(9),sh(0.4) ).

spring('LC-055J-7', od(0.720),hd(0.75),wd(0.055),
                load__sh(13.0),fl(2.0),k(8.5),sh(0.421) ).

spring('LC-055J-5', od(0.720),hd(0.75),wd(0.055),
                load__sh(13.0),fl(1.50),k(11.5),sh(0.331) ).

spring('LC-065J-5', od(0.720),hd(0.75),wd(0.065),
                load__sh(19.0),fl(1.50),k(19.0),sh(0.465) ).

spring('LC-063h-5', od(0.600),hd(0.625),wd(0.063),
                load__sh(23.0),fl(1.250),k(30.0),sh(0.457) ).

/*
```

SOLID GEOMETRY DEFINITIONS

```
*/

part(rod,[color(green)],P,data(
        Identity,
        elements(cylinder),
              [
                 radius(RAD),
                 center([0.0,0.0,0.0]),
                 length(LEN),
                 axis( X__unit )
              ]
        )) :-
        spec(identity(Identity)),

        param( rod(diameter(D)),P),
        param( rod(length(LEN)), P ),
        x__unit(X__unit),
        RAD is D / 2 , !.

part(spring1,[],P,
        data(
                rf(offset(X,0,0),angles(0,90,0)),
        elements(spring),
        []
        )) :-

        param(spring1(offset(X)),P),
        !.
```

```
she(spring,[color(red)],P,
        data(
        length(Spring__length), mean__radius(Mean__radius),
                pitch(Pitch),
        rf( offset(0,0,0), angles(0,0,0) ), /* was
offset(X,0,0) */
        curve( arc( s(Wire__start,0),c(Wire__center,0),
                e(Wire__start,0)))

        )) :-
        param(spring(length(LEN)),P),
        param(spring(id(ID)),P),
        param(spring(od(OD)),P),
        Mean__radius is ( ID + OD) / 4,
        Spring__length is LEN,
        Wire__dia is ( OD - ID)/2,
        Wire__start is Wire__dia / 2,
        Wire__center is 0.0,
        Pitch is 4.0,
        !.

part(spring2,[color(blue)],P,
        data(
                rf(offset(X,0,0),angles(0,90,0)),
        elements(spring),
        []
        )) :-

        param( spring2( offset(X) ),P),
        !.

part(bearing,[],P,
        data(
        rf( offset(0,0,0), angles(0,0,0) ),
        elements(tube),
        [ tube( id(ID), od(OD), len(LEN),
                center([ X, 0.0, 0.0]), axis(Xaxis)) ]
        )) :-
        x__unit( Xaxis),
        param(bearing(len(LEN)),P),
        param(bearing(id(ID)),P),
        param(bearing(od(OD)),P),
        param(center__x(X),P),
        !.

part(bearing2,[color(yellow)],P,
        data(
                rf(RFO,RFA),
        elements(bearing),
        [ center__x(X) ]
        )) :-
```

```
        spec(rf(RFO,RFA)),
        param( bearing2( offset(X) ),P),
        !.

part(bearing1,[color(yellow)],P,
        data(
                rf(RFO,RFA),
        elements(bearing),
        [ center__x(X) ]
        )) :-

        spec(rf(RFO,RFA)),
        param( bearing1( offset(X) ),P),
        !.

rev__closed( housing ,
        [ color(blue),
        comment(' The piston housing enclosing the mechanism')],
        Parameters,
        data(
                angle( 180 ),
                rf(OFFSET ,angles(0,0,180)),
                curve(
                        v(X1,Y4), v(X4,Y4), v(X4,Y1),
                        v(X3,Y1), v(X3,Y2), v(X2,Y2),
                        v(X2,Y3), v(X1,Y3), v(X1,Y4))
        )) :-

        param(
           housing(
              radii(
                 H__spring__radius, H__bearing__radius,
                 H__rodclear__radius, H__outside__radius),
              lens( H__springtravel , H__bearings, H__rod ),
              rf( OFFSET,ANGLES )),
           Parameters ),
        X1 is 0.0,
        X2 is H__springtravel,
        X3 is X2 + H__bearings,
        X4 is X3 + H__rod,

        Y1 is H__rodclear__radius,
        Y2 is H__bearing__radius,
        Y3 is H__spring__radius,
        Y4 is H__outside__radius, !.

sle(junk,__,__,__) :- !.

part(lvdt__cap,[color(cyan)],P,data(
        rf( Offset , Angles ),
        elements( cap ),[] )) :-
        param( lvdt__cap__rf( Offset, Angles ) , P),!.
```

```
part(internal__cap,[color(yellow)],P,data(
        rf( Offset , Angles ),
        elements( cap ),[] )) :-
        param( cap__rf( Offset, Angles ) , P),!.

sle( cap, [] , Params , data(
        delta( 0,0 ,Length ),
        rf( OFF , angles( 0,-(90),0)),
        curve( arc( s( X1,0), c( 0,0),e(X1,0)))
        )) :-

        spec(identity(rf(OFF,__))),
        param( cap( thickness(Cap__thickness),
                od( Cap__OD) ), Params),

        X1 is Cap__OD * 0.5,
        Length is Cap__thickness, !.
```

SPRING PISTON EXAMPLE

Table 1. Design Possibility 1.

rod					
length		4.4500			
rod					
diameter		0.3750			
rod					
offset		0.0			
spring1					
offset		0.0			
spring					
id		0.6100			
spring					
od		0.7200			
spring					
length		0.7875			
spring2					
offset		3.6625			
bearing					
id		0.3750			
bearing					
od		0.6250			
bearing					
len		0.8750			
bearing1					
offset		0.7875			
bearing2					
offset		2.5375			
housing					
radii		0.4750	0.3125	0.1975	0.7250
lens		2.375	2.6250	0.2500	
rf					
offset		-1.2500	0.0	0.0	
angles	0.0	0.0	0.0		
cap					
thickness		0.2500			
od		0.7500			
cap__rf					
offset		0.0	0.0	0.0	
angles		0.0	0.0	0.0	
lvdt__cap__rf					
offset		4.4500	0.0	0.0	
angles		0.0	180.0	0.0	

Table 1. Design Possibility 2.

rod				
length	4.6750			
rod				
diameter	0.3750			
rod				
offset	0.0			
spring1				
offset	0.0			
spring				
id	0.6100			
spring				
od	0.7200			
spring				
length	0.9000			
spring2				
offset	3.7750			
bearing				
id	0.3750			
bearing				
od	0.6250			
bearing				
len	0.8750			
bearing1				
offset	0.9000			
bearing2				
offset	2.6500			
housing				
radii	0.4750	0.3125	0.1975	0.7250
lens	2.1500	2.6250	0.2500	
rf				
offset	-1.2500	0.0	0.0	
angles	0.0	0.0	0.0	
cap				
thickness	0.2500			
od	0.7500			
cap__rf				
offset	0.0	0.0	0.0	
angles	0.0	0.0	0.0	
lvdt__cap__rf				
offset	4.6750	0.0	0.0	
angles	0.0	180.0	0.0	

Table 1. Design Possibility 3.

rod				
length	4.2250			
rod				
diameter	0.3750			
rod				
offset	0.0			
spring1				
offset	0.0			
spring				
id	0.6100			
spring				
od	0.7200			
spring				
length	0.6750			
spring2				
offset	3.5500			
bearing				
id	0.3750			
bearing				
od	0.6250			
bearing				
len	0.8750			
bearing1				
offset	0.6750			
bearing2				
offset	2.4250			
housing				
radii	0.4750	0.3125	0.1975	0.7250
lens	1.9250	2.6250	0.2500	
rf				
offset	-1.2500	0.0	0.0	
angles	0.0	0.0	0.0	
cap				
thickness	0.2500			
od	0.7500			
cap__rf				
offset	0.0	0.0	0.0	
angles	0.0	0.0	0.0	
lvdt__cap__rf				
offset	4.2250	0.0	0.0	
angles	0.0	180.0	0.0	

Table 1. Design Possibility 4.

rod				
length	4.2250			
rod				
diameter	0.3750			
rod				
offset	0.0			
spring1				
offset	0.0			
spring				
id	0.5900			
spring				
od	0.7200			
spring				
length	0.6750			
spring2				
offset	3.5500			
bearing				
id	0.3750			
bearing				
od	0.6250			
bearing				
len	0.8750			
bearing1				
offset	0.6750			
bearing2				
offset	2.4250			
housing				
radii	0.4750	0.3125	0.1975	0.7250
lens	1.9250	2.6250	0.2500	
rf				
offset	-1.2500	0.0	0.0	
angles	0.0	0.0	0.0	
cap				
thickness	0.2500			
od	0.7500			
cap__rf				
offset	0.0	0.0	0.0	
angles	0.0	0.0	0.0	
lvdt__cap__rf				
offset	4.2250	0.0	0.0	
angles	0.0	180.0	0.0	

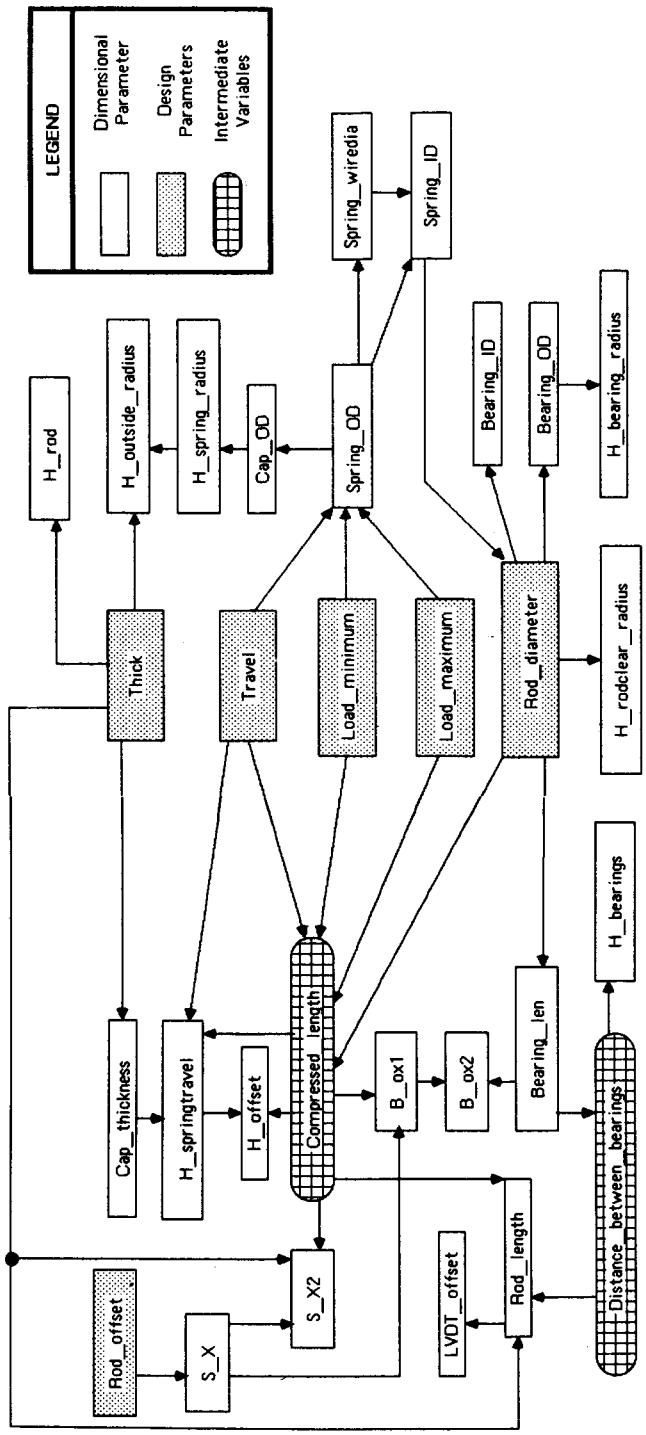


Figure 6. Dimensional Connectivity of Spring Piston.

SPRING PISTON EXAMPLE

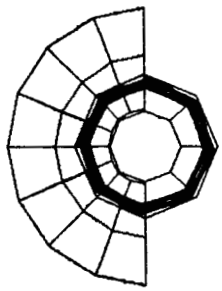


Figure 7. Spring Piston — Side View.

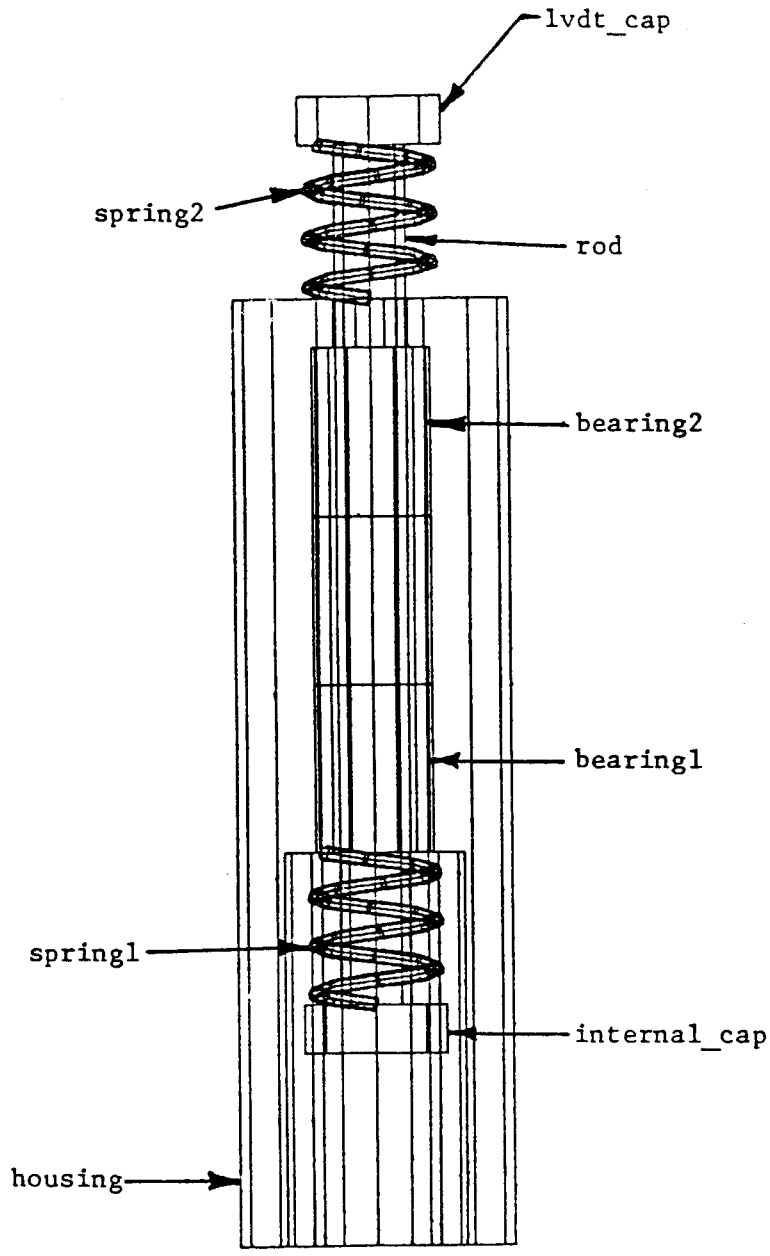


Figure 8. Spring Piston — Front View.

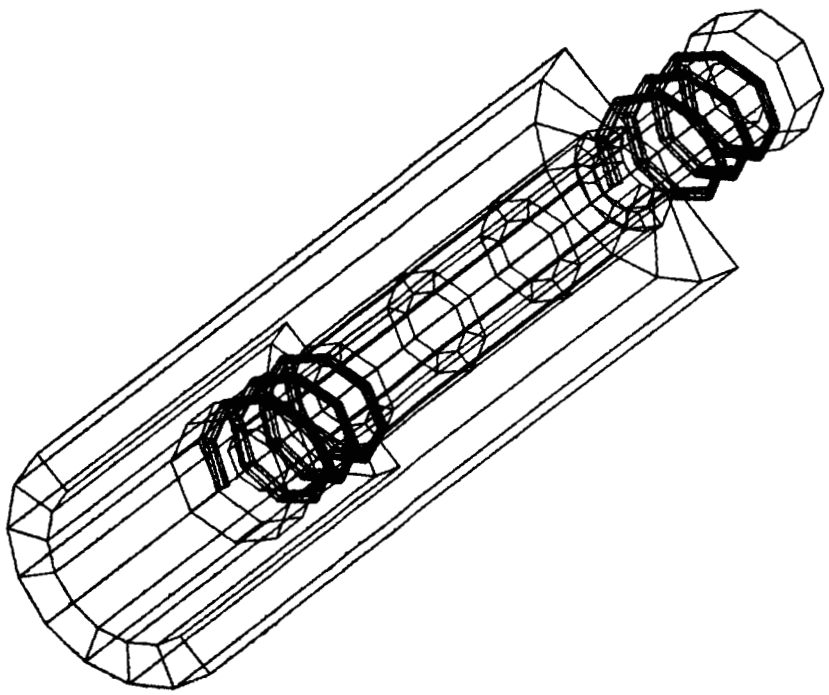


Figure 9. Spring Piston — Isometric View.

SPRING PISTON EXAMPLE
Redesigned Spring Piston Definition

```
/*
                    DESIGN SPECIFICATIONS
*/

                /*      At least this Travel        */
spec(travel(1.0)).
spec(load_max(30.0)).                                   /*      At solid height */
spec(load_min(10.0)).
spec(rf(offset(0.0,0.0,0.0),angles(0.0,0.0,0.0))).
spec(thickness(0.25)).
spec(identity(
        rf(offset(0.0,0.0,0.0),angles(0.0,0.0,0.0))
        )).
spec( rod,diameter(0.375)).
spec( rod,offset(0.0)).

/*
                    DESIGN CONSTRAINTS
*/

constraint( load , Load_at_solid_height, Max_load, Min_load ) :-
        Load_at_solid_height < Max_load,
        Load_at_solid_height > Min_load.

constraint( travel, Travel , Free_length , Solid_height ) :-
        Travel < Free_length - Solid_height.

constraint( rod_and_spring , Spring_ID, Rod_diameter ) :-
        Spring_ID > Rod_diameter.


design( piston , data(
        piston , elements(rod,bearing1,bearing2,
        internal_cap,lvdt_cap ,housing, spring1, spring2) )) :-
         !.

/*
                    PISTON DESIGN PREDICATE
*/
```

```
piston( [
        rod( length(Rod__length)),
        rod( diameter( Rod__diameter)),
        rod( offset( Rod__offset)),
        spring1( offset( S__X)),
        spring( id( Spring__ID)),
        spring( od( Spring__OD)),
        spring( length( Compressed__length)),
        spring2( offset( S__X2)),
        bearing( id( Bearing__ID)),
        bearing( od( Bearing__OD)),
        bearing( len( Bearing__len)),
        bearing1( offset( B__ox1)),
        bearing2( offset( B__ox2)),
        housing(
                radii(
                        H__spring__radius, H__bearing__radius,
                        H__rodclear__radius, H__outside__radius),
                lens( H__springtravel , H__bearings, H__rod ),
                rf( offset(H__offset,0,0) , angles(0,0,0))
                ),
        cap( thickness( Cap__thickness), od( Cap__OD) ),
        cap__rf( offset( 0,0,0), angles(0,0,0)),
        lvdt__cap__rf( offset(LVDT__offset,0,0) , angles(0,180,0) ),
        cap__spotface(Cap__spotface)

        ] ) :-

        spec( load__max( Load__maximum)),
        spec( load__min( Load__minimum)),
        spec( travel( Travel)),
        spec( thickness( Thick)),

        spec( rod,diameter(Rod__diameter)),
        spec( rod,offset(Rod__offset)),

        spring(Spring__name, od(Spring__OD),hd(Spring__hd),
                wd(Spring__wiredia),load__sh(Load__at__solid__height),
                fl(Free__length), k(Spring__K), sh(Solid__height) ),

/*      Find a valid spring         */

        constraint( load, Load__at__solid__height , Load__maximum
                                        , Load__minimum ),

        constraint( travel, Travel , Free__length , Solid__length ),

        ball__bushing(Type ,bore(Rod__diameter),od(Bearing__OD),
                        len(Bearing__len)),

        /* Rod must fit inside the spring */
```

```
        Spring__ID is Spring__OD - 2 * Spring__wiredia,
        constraint( rod__and__spring , Spring__ID, Rod__diameter ),

/*
                        +-----------------------------+
                        |  DEFINE PART ASSOCIATIVITY  |
                        +-----------------------------+
*/

        Compressed__length is Free__length * 0.45 ,
        Distance__between__bearings is 3 * Bearing__len,
        spotface__depth( Cap__spotface , Rod__diameter ),
        S__X is Rod__offset + Cap__spotface,

        S__X2 is S__X + Compressed__length +
Distance__between__bearings
                                        + Thick,

        Rod__length is
                Distance__between__bearings
                + (Compressed__length * 2.0) + Thick +
                2 * Cap__spotface ,

        Cap__thickness is Thick,
        Cap__thickness__OFFSET is 0 , /* Prolog quirk */
        Cap__OD is Spring__hd,

        Bearing__ID is Rod__diameter,
        B__ox1 is Compressed__length + S__X,
        B__ox2 is B__ox1 + 2.0 * Bearing__len,

        H__spring__radius is ( Cap__OD * 0.5 ) + 0.1,
        H__bearing__radius is Bearing__OD * 0.5,
        H__rodclear__radius is Rod__diameter * 0.5 + 0.010,

        H__springtravel is Cap__thickness + Compressed__length
                                                + Travel,
        H__bearings is Distance__between__bearings ,
        H__rod is Thick,
        H__outside__radius is H__spring__radius + Thick,
        H__offset is -(H__springtravel) + Compressed__length
                        + Cap__spotface,
        LVDT__offset is Rod__length.

/*
                        +-----------------------------+
                        |  LINEAR BEARING DATA BASE   |
                        |  THOMPSON LINEAR BEARING CO |
                        +-----------------------------+
*/

ball__bushing('XA-61014',bore(0.375),od(0.6250),len(0.875)).
```

```
/*
```

STOCK SPRING DATA BASE
LEE SPRING COMPANY

```
*/

spring('LC-055J-6', od(0.720),hd(0.75),wd(0.055),
                load__sh(13.0),fl(1.75),k(9),sh(0.4) ).

spring('LC-055J-7', od(0.720),hd(0.75),wd(0.055),
                load__sh(13.0),fl(2.0),k(8.5),sh(0.421) ).

spring('LC-055J-5', od(0.720),hd(0.75),wd(0.055),
                load__sh(13.0),fl(1.50),k(11.5),sh(0.331) ).

spring('LC-065J-5', od(0.720),hd(0.75),wd(0.065),
                load__sh(19.0),fl(1.50),k(19.0),sh(0.465) ).

spring('LC-063h-5', od(0.600),hd(0.625),wd(0.063),
                load__sh(23.0),fl(1.250),k(30.0),sh(0.457) ).

/*      Spot face definition      */

spotface__depth( SF , RD ) :-
        SF1 is RD / 4.0,
        /* Make it a fraction of eights. */
        make__integral__fraction( SF1 , 8.0, SF ),
        SF > 0, !.
spotface__depth( 0.125 , __ ) :-!.

Make__integral__fraction( R , FRAC, IF ) :-
        IF is float( round(R * FRAC)) / float(FRAC), !.

/*
```

SOLID GEOMETRY DEFINITIONS

```
*/

part(rod,[color(green)],P,data(
        Identity,
        elements(cylinder),
              [
                 radius(RAD),
                 center([0.0,0.0,0.0]),
                 length(LEN),
                 axis( X__unit )
              ]
        )) :-
        spec(identity(Identity)),
```

```
        param( rod(diameter(D)),P),
        param( rod(length(LEN)), P ),
        x__unit(X__unit),
        RAD is D / 2 , !.

part(spring1,[],P,
        data(
                rf(offset(X,0,0),angles(0,90,0)),
        elements(spring),
        []
        )) :-

        param(spring1(offset(X)),P),
        !.

she(spring,[color(red)],P,
        data(
        length(Spring__length), mean__radius(Mean__radius),
                pitch(Pitch),
        rf( offset(0,0,0), angles(0,0,0) ), /* was
offset(X,0,0) */
        curve( arc( s(Wire__start,0),c(Wire__center,0),
                e(Wire__start,0)))

        )) :-
        param(spring(length(LEN)),P),
        param(spring(id(ID)),P),
        param(spring(od(OD)),P),
        Mean__radius is ( ID + OD) / 4,
        Spring__length is LEN,
        Wire__dia is ( OD - ID)/2,
        Wire__start is Wire__dia / 2,
        Wire__center is 0.0,
        Pitch is 4.0,
        !.

part(spring2,[color(blue)],P,
        data(
                rf(offset(X,0,0),angles(0,90,0)),
        elements(spring),
        []
        )) :-
        param( spring2( offset(X) ),P),

        !.
part(bearing,[],P,
        data(
        rf( offset(0,0,0), angles(0,0,0) ),
        elements(tube),
        [ tube( id(ID), od(OD), len(LEN),
                center([ X, 0.0, 0.0]), axis(Xaxis)) ]
        )) :-
        x__unit( Xaxis),
```

```
        param(bearing(len(LEN)),P),
        param(bearing(id(ID)),P),
        param(bearing(od(OD)),P),
        param(center__x(X),P),
        !.

part(bearing2,[color(yellow)],P,
        data(
                rf(RFO,RFA),
        elements(bearing),
        [ center__x(X) ]
        )) :-
        spec(rf(RFO,RFA)),
        param( bearing2( offset(X) ),P),
        !.
part(bearing1,[color(yellow)],P,
        data(
                rf(RFO,RFA),
        elements(bearing),
        [ center__x(X) ]
        )) :-
        spec(rf(RFO,RFA)),
        param( bearing1( offset(X) ),P),
        !.

rev__closed( housing ,
        [ color(blue),
        comment(' The piston housing enclosing the mechanism')],
        Parameters,
        data(
                angle( 180 ),
                rf(OFFSET ,angles(0,0,180)),
                curve(
                        v(X1,Y4), v(X4,Y4), v(X4,Y1),
                        v(X3,Y1), v(X3,Y2), v(X2,Y2),
                        v(X2,Y3), v(X1,Y3), v(X1,Y4))
        )) :-

        param(
          housing(
            radii(
              H__spring__radius, H__bearing__radius,
              H__rodclear__radius, H__outside__radius),
            lens( H__springtravel , H__bearings, H__rod ),
            rf( OFFSET,ANGLES )),
          Parameters ),
        X1 is 0.0,
        X2 is H__springtravel,
        X3 is X2 + H__bearings,
        X4 is X3 + H__rod,
```

```
        Y1 is H__rodclear__radius,
        Y2 is H__bearing__radius,
        Y3 is H__spring__radius,
        Y4 is H__outside__radius, !.

sle(junk,__,__,__) :- !.

part(lvdt__cap,[color(cyan)],P,data(
        rf( Offset , Angles ),
        elements( cap ),[] )) :-
        param( lvdt__cap__rf( Offset, Angles ) , P),!.

part(internal__cap,[color(yellow)],P,data(
        rf( Offset , Angles ),
        elements( cap ),[] )) :-
        param( cap__rf( Offset, Angles ) , P),!.

rev__open( cap, [] , Params , data(
        angle(360.0),
        rf( OFF ,ANG),
        curve( v( X1,Y1), v(X1,Y2), v(X3,Y2), v(X3,Y4),
                v(X5,Y4), v(X5,Y1)
              )
        )) :-
        spec(identity(rf(OFF,ANG))),
        param( cap( thickness(Cap__thickness),
                od( Cap__OD) ), Params),
        param( cap__spotface(SP), Params),
        param( rod(diameter( Rod__dia)) , Params),

        X1 is 0.0,          Y1 is 0.0,
        X3 is SP ,          Y2 is Rod__dia * 0.5,
        X5 is SP - Cap__thickness ,          Y4 is Cap__OD * 0.5, !.
```

SPRING PISTON EXAMPLE

Table 2. Redesign Possibility 1.

rod				
length	*4.7000*			
rod				
diameter	0.3750			
rod				
offset	0.0			
spring1				
offset	*0.1250*			
spring				
id	0.6100			
spring				
od	0.7200			
spring				
length	0.7875			
spring2				
offset	*3.7875*			
bearing				
id	0.3750			
bearing				
od	0.6250			
bearing				
len	0.8750			
bearing1				
offset	*0.9125*			
bearing2				
offset	*2.6625*			
housing				
radii	0.4750	0.3125	0.1975	0.7250
lens	2.375	2.6250	0.2500	
rf				
offset	-1.2500	0.0	0.0	
angles	0.0	0.0	0.0	
cap				
thickness	0.2500			
od	0.7500			
cap__rf				
offset	0.0	0.0	0.0	
angles	0.0	0.0	0.0	
lvdt__cap__rf				
offset	*4.7000*	*0.0*	*0.0*	
angles	0.0	180.0	0.0	
cap__spotface	*0.1250*			

Table 2. Redesign Possibility 2.

rod				
length	*4.9250*			
rod				
diameter	0.3750			
rod				
offset	0.0			
spring1				
offset	*0.1250*			
spring				
id	0.6100			
spring				
od	0.7200			
spring				
length	0.9000			
spring2				
offset	*3.9000*			
bearing				
id	0.3750			
bearing				
od	0.6250			
bearing				
len	0.8750			
bearing1				
offset	*1.250*			
bearing2				
offset	*2.7750*			
housing				
radii	0.4750	0.3125	0.1975	0.7250
lens	2.1500	2.6250	0.2500	
rf				
offset	-1.2500	0.0	0.0	
angles	0.0	0.0	0.0	
cap				
thickness	0.2500			
od	0.7500			
cap__rf				
offset	0.0	0.0	0.0	
angles	0.0	0.0	0.0	
lvdt__cap__rf				
offset	*4.9250*	*0.0*	*0.0*	
angles	0.0	180.0	0.0	
cap__spotface	*0.1250*			

Table 2. Redesign Possibility 3.

rod				
length	*4.4750*			
rod				
diameter	0.3750			
rod				
offset	0.0			
spring1				
offset	*0.1250*			
spring				
id	0.6100			
spring				
od	0.7200			
spring				
length	0.6750			
spring2				
offset	*3.6750*			
bearing				
id	0.3750			
bearing				
od	0.6250			
bearing				
len	0.8750			
bearing1				
offset	*0.8000*			
bearing2				
offset	*2.5500*			
housing				
radii	0.4750	0.3125	0.1975	0.7250
lens	1.9250	2.6250	0.2500	
rf				
offset	-1.2500	0.0	0.0	
angles	0.0	0.0	0.0	
cap				
thickness	0.2500			
od	0.7500			
cap__rf				
offset	0.0	0.0	0.0	
angles	0.0	0.0	0.0	
lvdt__cap__rf				
offset	*4.4750*	*0.0*	*0.0*	
angles	0.0	180.0	0.0	
cap__spotface	*0.1250*			

Table 2. Redesign Possibility 4.

rod				
length	*4.4750*			
rod				
diameter	0.3750			
rod				
offset	0.0			
spring1				
offset	*0.1250*			
spring				
id	0.5900			
spring				
od	0.7200			
spring				
length	0.6750			
spring2				
offset	*3.6750*			
bearing				
id	0.3750			
bearing				
od	0.6250			
bearing				
len	0.8750			
bearing1				
offset	*0.8000*			
bearing2				
offset	*2.5500*			
housing				
radii	0.4750	0.3125	0.1975	0.7250
lens	1.9250	2.6250	0.2500	
rf				
offset	-1.2500	0.0	0.0	
angles	0.0	0.0	0.0	
cap				
thickness	0.2500			
od	0.7500			
cap__rf				
offset	0.0	0.0	0.0	
angles	0.0	0.0	0.0	
lvdt__cap__rf				
offset	*4.4750*	*0.0*	*0.0*	
angles	0.0	180.0	0.0	
cap__spotface	*0.1250*			

Table 2 REDESIGN POSSIBILITY

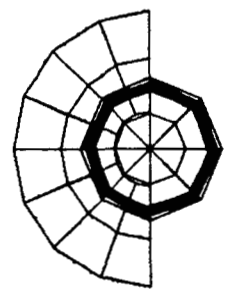


Figure 10. Redesigned Spring Piston — Side View.

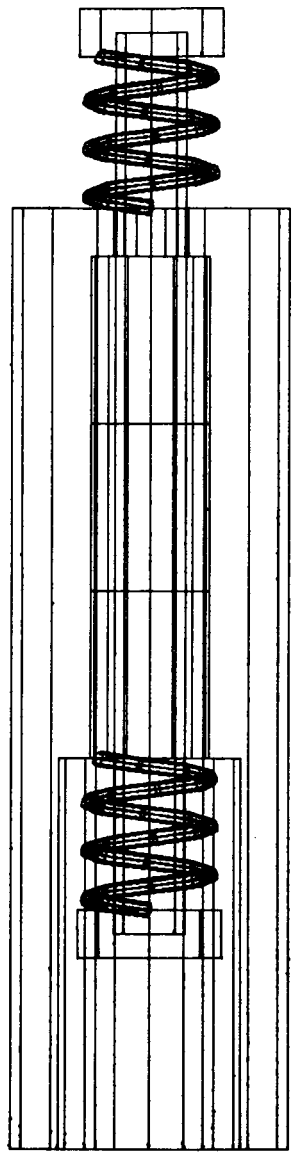


Figure 11. Redesigned Spring Piston — Front View.

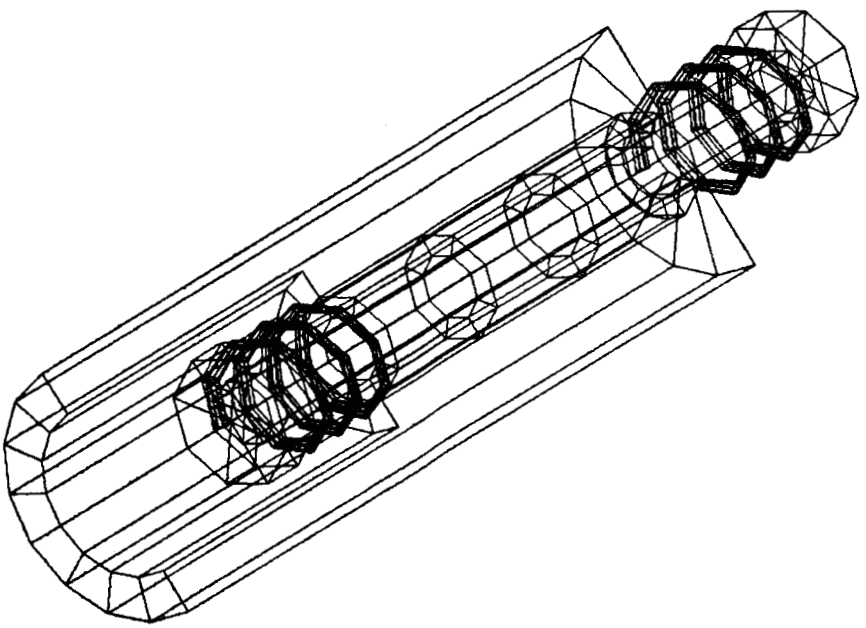


Figure 12. Redesigned Spring Piston — Isometric View.

APPENDIX C

PROLOG CAD SYSTEM CODE

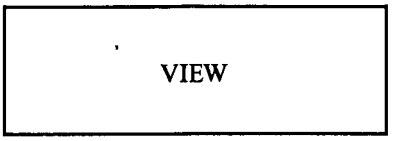
```
dl :- machine_code(31,[],_).
erase_dl :- machine_code(32,[],_).
init :- set(vt125), set(show(vertex)), set(width(40.0)).
cad :-
        consult('[ntp.cad.prolog]mth'),
        mth_init,
        consult('[ntp.cad.prolog]lib'),
        op(200,fx,set),
        assert(sys_cvue(0.0,0.0,0.0)),
        assert(sys_rvue(0.0,0.0,0.0)),
        assert(sys_width(100.0)),
        assert(sys_resolution(8)),
        assert(display_verify(_)),
        !.
/*
```

SET PREDICATES

```
*/
/*      The following arguments are valid for SET

Terminals
        vt125           Graphic output formatted for a VT125.
        lexidata        Graphic output formatted for a Lexidata Solid View.

Display
        resolution(N)   N is the number of line segments that approximate a curved surface.

        show(vertex)    Graphic output will display the brep as lines.
        show(patch)     Graphic output will display the brep as faces.

        clip(depth)     Turn on depth clipping
        clip(nodepth)   Turn off depth clipping ( just x and y ).

        cvue(X,Y,Z)     Set the center of the screen to X,Y,Z.
        rvue(R,P,Y)     Set the rotation of the screen by R,P,Y in degrees.
        width(W)        Set the width of the screen W units.

        [no]verify      Enable/Disable printing of design paramters used in a design.
*/

set(vt125) :- not(sys_terminal(vt125)),
                retractall(sys_terminal(_)),
                assert(sys_terminal(vt125)),
                (
                  ( sys_terminalinit(vt125),
                        machine_code(15,[],_) )
                  ;
```

```
                ( assert(sys__terminalinit(vt125)) ,
                        machine__code(10,[],__) )
                ),
                                view__window,!.

set(vt125) :- !.
set(lexidata) :- not(sys__terminal(lexidata)),
                retractall(sys__terminal(__)),
                assert(sys__terminal(lexidata)),
                set(show(patch)),
                (
                ( sys__terminalinit(lexidata),
                        machine__code(7,[],__) )
                 ;
                ( assert(sys__terminalinit(lexidata)) ,
                 machine__code(1,[],__) ) ),
                view__window, !.
set(lexidata) :- !.
set(la100) :- not(sys__terminal(la100)),
                retractall(sys__terminal(__)),
                assert(sys__terminal(la100)),
                machine__code(42,[],__),
                (
                ( sys__terminalinit(la100) )
                 ;
                ( assert(sys__terminalinit(la100)) ,
                 machine__code(40,[],__) ) ),
                view__window, !.

set(la100) :- !.

set(resolution(N)) :- N > 2, retract(sys__- resolution(__)),
                assert(sys__resolution(N)), !.
set(resolution(N)) :- nl, write(' Illegal resolution '),
                write(N), nl, break, abort.
set(show(vertex)) :-
        not(sys__show(vertex)) ,
        retractall(sys__show(__)),
        assert(sys__show(vertex)).

set(show(patch)) :-
        not(sys__show(patch)),
        retractall(sys__show(__)),
        assert(sys__show(patch)).

set(cvue(A,B,C)) :-
        X is float(A), Y is float(B), Z is float(C),
        retract(sys__cvue(__,__,__)),
        assert(sys__cvue(X,Y,Z)),
        view__window, !.
```

```
set(rvue(A,B,C)) :-
        X is float(A), Y is float(B), Z is float(C),
        retract(sys__rvue(__,__,__)),
        assert(sys__rvue(X,Y,Z)),
        view__window, !.

set(width(A)) :-
        X is float(A),
        retract(sys__width(__)),
        assert(sys__width(X)),
        view__window, !.
set(verify) :-
        retractall(display__verify(__)),
        assert(
                (display__verify(R) :-
                nl, display__verify1(R), !) ), !.
set(noverify) :-
        retractall(display__verify(__)),
        assert( display__verify(__) ), !.

set(clip(depth)) :- machine__code(31,[0.0],__), !.
set(clip(nodepth)) :- machine__code(31,[1.0],__), !.

view__window :-
        sys__width(WIDTH),
        sys__cvue(X,Y,Z),
        sys__rvue(Z__ANGLE,Y__ANGLE,X__ANGLE),
        machine__code(30,[WIDTH,X,Y,Z,Z__ANGLE,Y__ANGLE,X__ANGLE] ,__), !.

alphamode :- sys__terminal(vt125), machine__code(12,[],__), !.
alphamode :- sys__terminal(lexidata), !.
alphamode :- sys__terminal(la100), !.

erase :- sys__terminal(vt125), machine__code(11,[],__).
erase :- sys__terminal(lexidata), machine__code(6,[],__).
erase :- sys__terminal(la100), machine__code(43,[],__).

plot :- sys__terminal(la100), machine__code(41,[],__),!.
plot.

edit(A) :- machine__code(20,[A],__), !.

/*
```




```
*/
view :- sys__view(Name), view(Name), !.
view( Name ) :- retractall( sys__view(__)),
                assert( sys__view(Name)),
                view( Name , [] , 0), alphamode, !.
```

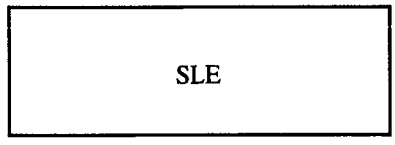
```
view( [First__name| Rest__of__names] , Params , Indent ) :-
        view( First__name, Params, Indent),
        view( Rest__of__names, Params , Indent ), !.
view( [] , __ , __ ) :- !.

view( Name , Params , Indent ) :-
                  view__design( Name , Params , Indent ),
                   alphamode, !.
view( Name , Params , Indent ) :-
                  view__part( Name , Params , Indent ),
                  alphamode, !.
view( Name , Params , Indent ) :-
                  view__sle( Name , Params , Indent ),
                  alphamode, !.
view( Name , Params , Indent ) :-
                  view__rev__open( Name , Params , Indent ),
                  alphamode, !.
view( Name , Params , Indent ) :-
                  view__rev__closed( Name , Params , Indent ),
                  alphamode, !.
view( Name , Params , Indent ) :-
                  view__she( Name , Params , Indent ),
                  alphamode, !.
view( Name , Params , Indent ), :-
        nl, write(' Definition not found: ' ) , write(Name),
         nl, restart, !.


/*
```

DESIGN

```
*/
view__design( Name , __ , Ident ) :-
        design( Name , data( Design__function , Elements ) ),
        display__name( design, Ident, Name),
        Elements =.. [elements|Names],
        view__all__designs( Names , Design__function , Ident ), !.

view__all__designs( Names , Design__function , Ident ) :-

        DF =.. [Design__function,Params],
        call(DF),
        display__verify(Params),
        Ident1 is Ident + 2,
        view( Names , Params, Ident1 ),
        nl,
        break,
        fail, !.
view__all__designs( __ , __, __ ) :-
        nl, write( ' Design options exhausted ' ),nl, restart, !.
```

```
/*
```


```
*/

view__part( Name , Parameters , Indent ) :-
        part( Name , Attribute__list , Parameters , Data),
        parse__attributes( Attribute__list ) ,
        display__name( part, Indent, Name),
        Data =.. [data,rf(Offset,Angles), Elements, E__params ],
        Elements=.. [ elements|Element__list ],
        append(E__params, Parameters , P3),

        /* Append the new transformation matrix */

                mth__trnsmat( Offset, Angles, Matrix),
                sys__transformation( Sys__matrix),
                mth__append__trnsfrm( Matrix, Sys__matrix,
                        New__matrix),
                asserta( sys__transformation( New__matrix)),
                !, trimcore,
        /* Now view each element */

                Indent1 is Indent + 2,
                view( Element__list , P3 , Indent1 ),

        /* Now remove the rf. and retract it. */
                retract(sys__transformation(New__matrix)), !.


/*
```




```
*/
view__sle( Name , Parameters , Indent ) :-
        sle( Name , Attribute__list , Parameters ,
                data(Delta,rf(Offset,Angles),Curve)),
        display__name( sle, Indent , Name),
        parse__attributes( Attribute__list ) ,
        Delta =.. [delta|Delta__vector],
        mth__trnsmat(Offset,Angles,Matrix),
        sys__transformation(Sys__matrix),
        mth__append__trnsfrm(Matrix,Sys__matrix,New__matrix),
```

```
        gmtr__curve__to__vertex( Curve , Verticies__2d),
        New__matrix = [New__matrix__R,__],
        mth__mul__matvec(New__matrix__R,Delta__vector,
                        Delta__vector__trn),
        display__make__sle(Verticies__2d , New__matrix,
                        Delta__vector__trn ),!.

display__make__sle(Vertex , Matrix , Delta__vector__trn ) :-
        gmtr__verts2d__3d__z(Vertex , Verts3d),
        mth__trnsfrm(Matrix , Verts3d , Vt1 ),
        mth__add__cl(Delta__vector__trn,Vt1,Vt2),
        display__make__sle1(Vt1,Vt2), !.

        /* Display Verticies */

display__make__sle1( Bottom__verts , Top__verts ) :-
        sys__show(vertex),
        display__make__sle__vertex(Bottom__verts , Top__verts), !.

display__make__sle__vertex([F1,V1|R1],[F2,V2|R2] ) :-
        display__verticies([F1,V1,V2,F2]),
        display__make__sle__vertex([V1|R1] , [V2|R2] ), !.
display__make__sle__vertex( [__], [__] ) :- !.

        /* Display Patches */

display__make__sle1( Bottom__verts , Top__verts ) :-
        sys__show(patch),
        display__make__sle__patch(Bottom__verts , Top__verts), !.

display__make__sle__patch( Bottom__verts , Top__verts ) :-
        display__patch(Bottom__verts),
                /* Save the first vertex */
        display__make__sle__patch1(Bottom__verts , Top__verts ),
        reverse(Top__verts , Top),
        display__patch(Top),!.

display__make__sle__patch1([F1,V1|R1],[F2,V2|R2] ) :-
        display__patch([F1,F2,V2,V1]),
        display__make__sle__patch1([V1|R1] , [V2|R2] ), !.
display__make__sle__patch1( [__], [__] ) :- !.
```

```
/*
```

REV__CLOSED

```
*/
view__rev__closed( Name , Parameters , Indent ) :-
        rev__closed( Name , Attribute__list , Parameters ,
        data(angle(Angle) , rf(Offset,Angles) , Curve)),

        display__name( rev__closed, Indent , Name ),
        parse__attributes( Attribute__list ) ,

        Angle__rad is Angle * 0.01745329252,
        mth__trnsmat(Offset,Angles,Matrix),
        sys__transformation(Sys__matrix),
        mth__append__trnsfrm(Matrix,Sys__matrix,New__matrix),

        gmtr__curve__to__vertex( Curve , Verticies__2d),
        display__make__revclosed( Verticies__2d , New__- matrix,
                                  Angle__rad ),
        display__make__revclosed__angles( Angle , Verticies__2d,
                                  New__matrix),

        !.

display__make__revclosed( [[Xvert,Yvert]|Rvert] , Matrix , Angle )
                :-

        sys__resolution(Res),
        Count is Res + 1,
        Angle__increment is Angle / Res,
        gmtr__arcverts1(
                Yvert,      /* The radius */
                0.0 ,       /* X center */
                0.0 ,       /* Y Center */
                0.0 ,       /* Start angle */
                Angle__increment ,
                Count ,     /* Count for loop*/
                Verts ),

gmtr__verts2d__3d__zyx(Xvert,Verts,Verts1),
mth__trnsfrm( Matrix , Verts1 , Verts__world ),
display__make__revclosed__1( Verts__world , Rvert , Matrix ,
                Angle__increment , Count ).

display__make__revclosed__1( V1 , [[Xvert,Yvert]|Rvert] , Matrix
                        , Angle__increment , Count ) :-
```

```
        gmtr__arcverts1(
                Yvert,      /* The radius */
                0.0 ,       /* X center */
                0.0 ,       /* Y Center */
                0.0 ,       /* Start angle */
                Angle__increment ,
                Count ,     /* Count for loop*/
                Verts ),
        gmtr__verts2d__3d__zyx(Xvert,Verts,Verts1),
        mth__trnsfrm( Matrix , Verts1 , V2 ),
        display__make__genrev__quad( V1 , V2 ),
        display__make__revclosed__1( V2 , Rvert , Matrix ,
                        Angle__increment , Count ), !.
display__make__revclosed__1(__,[],__,__,__ ) :- !.

display__make__revclosed__angles( Angle , __ , __ ) :-
        0.1E-8 > abs(Angle - 360.0) , !.
display__make__revclosed__angles( Angle , Verticies__2d, Matrix ) :-
        sys__show(vertex),
        dmrvat( Angle, Matrix , [Verticies__2d] ), !.

display__make__revclosed__angles( Angle , Verticies__2d, Matrix ) :-
        reverse( Verticies__2d, RV2D ),
        gmtr__make__curve__convex( RV2D, RV2D , TRIANGLES ),
        dmrvat( Angle, Matrix , TRIANGLES ), !.
dmrvat( __ , __ , [] ) :- !.
dmrvat( Angle, Matrix , [Verticies__2d|RV2D] ) :-
        gmtr__verts2d__3d__z( Verticies__2d, V3d),
        mth__trnsfrm( Matrix , V3d , V3d__a ),
        display__ngon( V3d__a),
        mth__trnsmat(offset(0,0,0),angles(0,0,Angle),Matrix1),
        mth__append__trnsfrm(Matrix1,Matrix, New__matrix),
        mth__trnsfrm( New__matrix, V3d , V3d__b ),
        reverse( V3d__b, V3DB ),
        display__ngon( V3DB ),
        dmrvat( Angle, Matrix, RV2D ),
        !.

/*
```

SOLID OF HELICAL EXTRUSION

```
*/
view__she( Name , Parameters , Indent ) :-

        she( Name , Attribute__list , Parameters ,
          data(
                        length(Length),
                        mean__radius(Mean__radius),
                        pitch(Pitch),
                        rf(Offset,Angles),
                        Curve)
        ),
```

```
        display__name( she, Indent , Name),
        parse__attributes( Attribute__list ) ,

        mth__trnsmat( Offset , Angles , Matrix ),
        sys__transformation(Sys__matrix),
        mth__append__trnsfrm( Matrix, Sys__matrix, Global__mat- rix),

        sys__resolution(Resolution),

        mth__pi(Pi),
        /* Pitch is [turns/inch] */
        /* Number of radians */
        No__of__turns is Pitch * Length * Pi * 2.0,

        Count is round( Pitch * Length * Resolution),
        Angle__increment is No__of__turns / Count,

        Z__inc is 1.0 / ( 2 * Pi * Pitch ),

        X is Mean__radius * cos(Angle__increment),
        Y is Mean__radius * sin(Angle__increment),
        Z is Z__inc * Angle__increment,
        Angle__inc__degree is Angle__increment * 180.0 / Pi,

        X__delta is X - Mean__radius,
        VNR = [ X__delta, Y , Z ],

        mth__trnsmat( offset( 0 , 0, Z),
                angles( Angle__inc__degree ,0,0) , Spin__matrix ),
/*                The angles needed to rotate the curve */
/*                perpendicular to the helix direction */
        mth__axis__angles__z( VNR, Helix__angles ),

                /* Convert bounding curve to verticies */
        gmtr__curve__to__vertex( Curve , Bounding__verticies ), !,
                /* Set the z value to be 0.0 */
        gmtr__verts2d__3d__z( Bounding__verticies ,
                                Bounding__verticies__3D ), !,
/*move the curve over to the helix*/
        Ang =.. [angles|Helix__angles ],

        mth__trnsmat( offset( Mean__radius , 0, 0),
                                Ang , Helix__matrix ),

        mth__trnsfrm( Helix__matrix, Bounding__vertici-□es__3D ,
                                Bverts__normal ), !,
        mth__trnsfrm( Global__matrix, Bverts__normal, BVF ),
        display__ngon( BVF ), !,

        display__make__she( Count, BVF, Bverts__normal ,
                Spin__matrix , Global__matrix),
```

```
        !.
display_make_she( 0,V,_,_,_) :- reverse( V,VR),
                                display_ngon( VR ), !.
display_make_she( Cur , V_OLD, V , M ,GM ) :-
        Cur1 is Cur-1,
        mth_trnsfrm( M , V , VP), !,
        mth_trnsfrm( GM , VP, VG), !,
        display_make_genrev_quad( V_OLD, VG ),
        display_make_she( Cur1, VG, VP , M,GM ), !.


/*
```

REV_OPEN

```
*/
view_rev_open( Name , Parameters , Indent ) :-
        rev_open( Name , Attribute_list , Parameters ,
             data( angle(Angle) , rf(Offset,Angles) , Curve) ),
        display_name( rev_open, Indent , Name),
        parse_attributes( Attribute_list ) ,

        Angle_rad is Angle * 0.01745329252,

        mth_trnsmat(Offset,Angles,Matrix),
        sys_transformation(Sys_matrix),
        mth_append_trnsfrm(Matrix,Sys_matrix,New_matrix),
        gmtr_curve_to_vertex( Curve , Verticies_2d),
        display_make_revopen( Verticies_2d , New_matrix,
                                        Angle_rad ),!.

display_make_revopen( [ [X,Y] , [Xvert,Yvert] | Rvert ] ,
                                Matrix , Angle ) :-
        sys_resolution(Res),
        Count is Res + 1,
        Angle_increment is Angle / Res,
        mth_trnsfrm( Matrix , [X,Y,0.0] , Vcenter ),
        gmtr_arcverts1(
                Yvert,      /* The radius */
                0.0 ,       /* X center */
                0.0 ,        /* Y Center */
                0.0 ,       /* Start angle */
                Angle_increment ,
                Count ,     /* Count for loop*/
                Verts ),
        gmtr_verts2d_3d_zyx(Xvert,Verts,Verts1),
        mth_trnsfrm( Matrix , Verts1 , V1),
        display_make_revopen_start( Vcenter, V1 ),
        display_make_revopen1( V1 , Rvert , Matrix ,
                                Angle_increment , Count )
        , !.
```

```
display__make__revopen1( V2 , [[X,Y]], Matrix , __ , __ ) :-
        mth__trnsfrm( Matrix , [X,Y,0.0] , V1__world ),
        display__make__revopen__end( V2 , V1__world ), !.

display__make__revopen1( V1 , [[Xvert,Yvert]|Rvert] , Matrix
                        , Angle__increment , Count ) :-

        gmtr__arcverts1(
                Yvert,      /* The radius */
                0.0 ,       /* X center */
                0.0 ,       /* Y Center */
                0.0 ,       /* Start angle */
                Angle__increment ,
                Count ,     /* Count for loop*/
                Verts ),
        gmtr__verts2d__3d__zyx(Xvert,Verts,Verts1),
        mth__trnsfrm( Matrix , Verts1 , V2 ),
        display__make__genrev__quad( V1 , V2 ),
        display__make__revopen1( V2 , Rvert , Matrix ,
                                Angle__increment , Count ), !.

/*                                  Display the triangular end patches                                  */
display__make__revopen__end( VS , VC ) :-
        sys__show(patch),
        display__make__revopen__patchm( VS, VC), !.
display__make__revopen__end( VS , VC ) :-
        sys__show(vertex),
        display__make__revopen__vertexm( VS, VC), !.

display__make__revopen__patchm( [V1,V2|RV] , VC ) :-
        display__patch([V1,V2,VC]),
        display__make__revopen__patchm([V2|RV] , VC ), !.
display__make__revopen__patchm( [__] , __ ) :- !, trimcore.

display__make__revopen__vertexm( [V1,V2|RV] , VC ) :-
        display__verticies([V1,V2,VC]),□        display__make__revopen__vertexm([V2|RV] , VC ), !.
display__make__revopen__vertexm( [__] , __ ) :- !, trimcore.

/*                                  Display the triangular Start patches                                */
display__make__revopen__start( VC , VS ) :-
        sys__show(patch),
        display__make__revopen__patch( VC , VS ) , !.

display__make__revopen__start( VC , VS ) :-
        sys__show(vertex),
        display__make__revopen__vertex( VC , VS ) , !.

display__make__revopen__patch( V__center , [V1,V2|Rv] ) :-
        display__patch([V__center , V1,V2]),
        display__make__revopen__patch( V__center , [V2|Rv] ), !.
display__make__revopen__patch( __ , [__] ) :- !.
```

```
display__make__revopen__vertex( V__center , [V1,V2|Rv] ) :-
        display__verticies([V__center , V1,V2]),
        display__make__revopen__vertex( V__center , [V2|Rv] ), !.
display__make__revopen__vertex( __ , [__] ) :- !.

/*
```

Misc Definitions

```
/*                          Define the initial transformation matrix                    */
sys__transformation([
                [1.0,0.0,0.0,
                 0.0,1.0,0.0,                          /* 3 x 3 rotation */
                 0.0,0.0,1.0],
                [0.0,0.0,0.0]]).                         /* Offset */
```

BIBLIOGRAPHIC DATA SHEET

1. Report No. NASA TM-86241	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle A RULE BASED COMPUTER AIDED DESIGN SYSTEM		5. Report Date October 1986 6. Performing Organization Code
7. Author(s) Timothy Premack		8. Performing Organization Report No. 86B0078
9. Performing Organization Name and Address Mechanical Engineering Branch (Code 731) NASA Goddard Space Flight Center Greenbelt, MD 20771		10. Work Unit No. 11. Contract or Grant No. 13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		14. Sponsoring Agency Code

15. Supplementary Notes

Originally submitted to the Johns Hopkins University as a masters essay.

16. Abstract

A Computer Aided Design (CAD) system is presented which supports the iterative process of design, the dimensional continuity between mating parts, and the hierarchical structure of the parts in their assembled configuration. Prolog, an interactive logic programming language, is used to represent and interpret the data base. The solid geometry representing the parts is defined in parameterized form using the swept volume method. The system is demonstrated with a design of a spring piston.

17. Key Words (Selected by Author(s))	18. Distribution Statement
Computer Aided Design, Design, Logic Programming, Prolog	Unclassified-Unlimited Subject Category - 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	62	A04

*For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1986