AUTOMATIC PLANNING OF ROBOTIC ASSEMBLY TASKS

Rukmini Vijaykumar

COINS Technical Report 88-75

September 1988

Laboratory for Perceptual Robotics
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

This research was supported by the National Science Foundation under grant number 87-12189 (Michael A. Arbib, Principal Investigator).

Author's present address: Department of Computer Science, University of Southern Maine, Portland, ME 04103.

AUTOMATIC PLANNING OF ROBOTIC ASSEMBLY TASKS

A Dissertation Presented

By

RUKMINI VIJAYKUMAR

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 1988

Department of Computer and Information Science

This research was supported by the National Science Foundation under grant number 87-12189 (M.A.Arbib, Principal Investigator).

AUTOMATIC PLANNING OF ROBOTIC ASSEMBLY TASKS

A Dissertation Presented

by

RUKMINI VIJAYKUMAR

Approved as to style and content by:

Michael A. Arbib, Chairperson

Robin J. Popplestone, Member

Daniel D. Corkill, Member

John R. Dixon, Outside Member

W. Richards Adrion, Department Chair
Computer and Information Science

ACKNOWLEDGEMENTS

The process of working on my dissertation has been long and arduous, and innumerable friends and colleagues have provided me with timely encouragement and emotional support. I owe much thanks to all of them. My deepest debt of gratitude is to the chairperson of my committee, Professor Michael Arbib, who set me off in this direction, and provided me with the necessary intellectual, moral, and financial support for completing this project. His consistent encouragement, patience, sensitivity, and the freedom he allowed me in developing the ideas in this dissertation, have been invaluable.

I wish to thank Professor Robin Popplestone for several stimulating discussions, his encouragement, support, and comraderie. This work benefited greatly from the software he made available in the UMass environment. His help in developing the interface with the RAPT system, and his patience and availability in answering my questions, are deeply appreciated. I wish to thank Dr. Daniel Corkill for his involvement in the project from its inception, his help with the application and analysis of the planning work in this dissertation, and for his encouragement and support in the course of this effort. I would like to thank Prof. John Dixon for introducing me to the ideas on symmetry, for his interest in this dissertation, and for serving on my committee. I also wish to thank Professors Robbie Moll, Lori Clarke, and Jack Wileden for providing me with advisory and financial support during the early part of my stay at UMass.

My fellow members of the Laboratory for Perceptual Robotics have been a great source of intellectual support. From them, I learnt about the complexities of the vast and interdisciplinary field of robotics that I had stumbled into. Special thanks go to Damian Lyons for his willingness to listen to my ideas, and for his valuable feedback on those that I committed to paper. Our many fruitful discussions, and collaborative efforts, have influenced this work. Specifically, the discussion on the overall system design in Chapter 3

evolved out of our collaborative efforts. I wish to thank Yanxi Liu for discussions during the early phase of this work. Thanks are also due to T.V. Subramaniam, Gordon Dakin, Thea Iberall, Judy Franklin, and Gerry Pocock, who provided an energetic and lively environment to work in. I wish to thank Miles Lane and the RCF community for systems support and for the impressive array of reliable resources that were made available for use.

I wish to thank my friends Thea Iberall, Brian Burns, and Debbi Strahman for providing me with a forum of accountability. It was great to have a place where I could vent my frustrations, express my confusions, make peace with myself when I faltered, and get back on track rapidly. They kept me focussed on "getting it done". I also wish to thank my friends Mary Schatzkamer, Anandan, and Debby Servi for their love and support through some of the more strenuous times.

I would like to thank my mother, my brothers Balu and Kalyan, and my sister Saras, who made countless sacrifices to enable me to pursue my dreams and gave me much love and encouragement thorough it all. I also wish to thank my friends Nalini, Gopi and Mary who put me up, and put up with me, through the final stages of writing this dissertation. Last, but not least, I would like to thank Vijay who was very much a part of my decision to undertake this project, and the challenges I met along the way, for his encouragement and support, and for always believing that I could do it.

object model is a hierarchical decomposition approach to the assembly planning problem, where the problem is refined in increasing levels of detail, first in terms of features that will be related by the assembly, and further, in terms of the spatial relationships that will arise between the surfaces that constitute the features. The reasoning processes that effect the problem refinement are based on the geometry of assembly operations and objects, constraints imposed by the sequencing of operations, and considerations of spatial consistency given the locality of features on objects.

An implementation of the object models, the decomposition approach, and the related reasoning processes exists in the program ASSEMBLE. The dissertation provides a description of the system as well as the results obtained from using ASSEMBLE on a set of experimental assembly tasks. Possible directions for extending the work are outlined. Significant among these is the incorporation of the analysis of volumetric interference between objects as another reasoning component of the system.

ABSTRACT

AUTOMATIC PLANNING OF ROBOTIC ASSEMBLY TASKS

September 1988

RUKMINI VIJAYKUMAR

B.Sc., Madras University

M.Sc., Indian Institute of Technology, Madras

M.Tech., Indian Institute of Technology, Madras

Ph.D., University of Massachusetts

Directed by: Professor Michael A. Arbib

There has been a growing interest in the field of robotics over the last several years. As the capabilities of the basic underlying mechanisms increase, efforts have intensified to endow such mechanisms with the ability to exhibit intelligent behavior. The notion of building robotic systems that can reason about the tasks they are required to execute while operating in incompletely specified, and uncertain environments continues to be an enticing, yet elusive, goal. Towards this goal, this dissertation presents a system architecture suitable for intelligent robotics and, within this framework, addresses the problem of planning assembly tasks.

The specific problem addressed in this dissertation is one where given a sequence of high-level assembly instructions, and selective information about the objects involved in the assembly, a final assembly configuration is to be determined based on which commands to a robot system that would realize the configuration can be generated.

A hierarchical object model is developed where objects are viewed as collections of features, and features, in turn, are viewed as collections of surfaces. Coupled with the

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

The general aim of robotics can be described as the mechanization of the basic *operative* intelligence people use in locating and manipulating objects. There are three principal aspects to this problem that deserve and receive attention, namely, *sensory interpretation*, *manipulation* and *reasoning*. Research in the *sensory* aspects of the problem attempts to organize raw data obtained from the sensors into perceptually meaningful units and to interpret such data into perceptually meaningful events. Current research in *manipulation* attempts to develop strategies required to control bodies moving through 3-dimensional space in situations where contact with the environment is undesirable as well as in situations where contact with the environment is necessary. The *reasoning* or *planning* component of the problem attempts to combine the sensory and manipulative capabilities of a mechanism in an effort to accomplish a specific goal. This dissertation is concerned with the problem of automating the reasoning required to accomplish a specified task. Specifically, it addresses the problem of planning robotic assembly tasks.

1.1 Task Planning

In the field of robotics, the subarea that concerns itself most with the reasoning aspects of the problem is the area of *task planning*. Task planning represents the ability to define a task in terms of desired relationships between a set of objects and to have a system automatically generate the necessary robot-level commands.

Task-level specifications typically consist of a sequence of instructions where each instruction includes the description of an operation and the specification of the objects

involved in the operation as in LAMA [31] and AUTOPASS [28]. Examples of task-level specifications from these proposed task-level languages are:

Insert the piston pin partway into the piston or

Screw the bracket and the interlock together.

Alternately, task-level specifications may be expressed purely in terms of the relationships between features of objects as in RAPT [45]. For example, $Face_1$ of $Object_1$ against $Face_2$ of $Object_2$ is a typical assembly specification statement in RAPT. Robot-level specifications, on the other hand, describe actions in terms of the necessary manipulator movements and sensing commands. The problem of transforming task-level specifications into robot-level specifications subsumes several significant problems in robotics and opens up problem areas on which little work has been done.

In order to break down a specified task into its constituent robot-level commands, several subproblems need to be solved. Significant among these are problems of planning a grasp, planning gross motions for transferring objects from one location to another, and planning fine motions. Grasp planning involves the selection of grasp site and end-effector configuration suitable for a task. Gross-motion planning requires the identification of path and trajectory parameters that avoid collision with other objects in the workspace. Fine-motion planning requires the selection of appropriate control strategies for effecting relative motions of objects in contact, that may be required to complete a task. The subproblems of grasp planning, gross-motion planning and fine-motion planning have been worked on to varying degrees [30,23,24,32,9,11,58,16,34,38]. However, no system exists today that integrates these to provide a solution to the task planning problem. A recent system, Handey [36], developed at MIT, is the most advanced task planning system to date. It combines object localization, collision free path planning, and grasping in one integrated system.

An essential characteristic of a task planning system is the ability to represent and reason about objects. Suitable representations of objects have been sought and developed for various purposes: for display in the graphics domain, for modeling and interpretation in computer vision, for design analysis in CAD systems. Robotics requires us to

extend or adapt these existing representations to facilitate reasoning about manipulation. Most of the existing representations for objects, across the various domains listed above, emphasize completeness of the representation and are consequently too detailed and inefficient for the purposes of reasoning about objects. In our view, the problem of planning a task requires the ability to support different, yet consistent, representations of the environment, events occurring in the environment, and actions the robot can perform. The representations may occur at varying levels of abstraction; and the required transformations are not only of the actions but also of the environmental representations and events.

Given the inherent uncertainty in the domain of robotics, any proposed approach to task planning needs to incorporate execution monitoring and replanning as well. This dissertation addresses the problem of planning robotic assembly tasks, focusing on the issue of representing and reasoning about objects. Issues relating to motion planning, execution monitoring, and replanning have influenced the design of the overall system architecture but are not directly addressed in this study.

1.2 Planning Assembly Tasks

Of specific interest to the field of robotics is the problem of robotic assembly. This interest stems from the potential for application of this evolving technology in industry as well as from the challenges it provides in terms of the precision it requires. Thus, solving the task planning problem in the domain of assembly is of great relevance to the field of robotics.

The term *assembly planning* has been used in the literature to mean a variety of problems. On the one hand, it refers to the problem of scheduling assembly operations where the order in which parts may arrive at the workstation could vary. Here the specific relationships that the objects are to hold with respect to each other is given; however, the order in which these relations are to be accomplished is not completely specified and constitutes the problem to be solved [52,27]. On the other hand, the problem of arriving at the manipulator-level commands given a sequence of assembly operations or

an assembly plan is addressed in [30]. In this case, the assembly is specified both in terms of the order in which the operations are to be performed as well as in terms of the specific relations the objects are supposed to hold. The problem is one of identifying the sequence of manipulator-level primitives that will accomplish these.

At either end of the spectrum, description of an assembly task takes the form of specifying in complete detail the relative pose the objects assume with respect to each other, typically as a homogeneous transformation matrix. The transformation matrix would describe the position and orientation of a coordinate frame embedded in one object relative to a coordinate frame in another object or the workspace. The problem of deriving the relative pose of objects from knowledge about the assembly operations and the objects involved in the assembly has not been addressed before.[1] This problem of (a) identifying the specific configuration of the resulting assembly based on object and task information, and (b) generating the robot-level commands to realize the configuration, is termed *assembly planning*. The work described in this dissertation is an attempt to develop a solution to the assembly planning problem.

1.3 Problem Statement

The problem addressed in this dissertation can be simply stated as follows:

> *Given a set of objects, information pertaining to the objects relevant for assembly, and a set of assembly operations that relates these objects, determine the pose of objects relative to each other that defines the exact assembly configuration, and the sequence of robot-level commands that realizes this configuration.*

The assembly planning problem as stated above raises the following questions:

[1] The RAPT system [46] derives the relative positions of objects based on the specification of a set of symbolic spatial relationships between features of objects. The work described in this dissertation operates from higher-level specifications of assembly tasks and derives RAPT level specifications as an intermediate representation of the task.

1. What information about an object is pertinent for the purposes of assembly?
2. How should we view the process of assembly in order to successfully derive the relative pose information?
3. What constitutes the set of assembly operations?
4. How do we define the robot-level commands?

The study described in this dissertation focuses on developing answers to the first two questions. Choices made in response to the latter questions serve to delimit the scope of this study.

1.4 Overview of the Dissertation

The two primary issues addressed in this dissertation are:

- The development of an abstract model of objects that would allow the explicit representation of object characteristics relevant for reasoning about assembly.
- The identification of the necessary reasoning processes and the control of their application for solving the assembly planning problem.

Both of these issues are addressed through building a program ASSEMBLE which attempts to solve the assembly planning problem. In addition, a research framework within which the assembly planning problem could be addressed has been developed.

There are two goals that underlie the efforts toward the development of the framework. One is that ASSEMBLE should be a component of an intelligent robot system that can be realistically integrated with other components. To this end, a system architecture has been developed, which relates a hierarchy of control units, motion planning components, sensory preprocessors, and a central database of information about the environment and the robot. The complexity of such a system, along with the uncertainty regarding actions producing intended effects, has led to a design of a system architecture that emphasizes hierarchical structuring and feedback between levels. The architecture, in this respect,

is similar in principle to the NBS system due to Albus, *et al.* [3] but different in scope. Placing ASSEMBLE in this context, allows us to define the appropriate level of input and output for the system.

The second goal is the separation of the device-independent aspects of planning an assembly task from the device-dependent aspects. Automatic planning of assembly tasks suitable for execution by a robot requires analysis of constraints arising from task and object geometry as well as constraints arising from the specific configuration of the robot system. The ASSEMBLE system confines its attention to constraints arising from task and object characteristics. Thus, ASSEMBLE does not directly deal with the problems of motion planning but interfaces with subsystems that would carry out the motion planning. It provides some of the task-related information that is pertinent for such motion planning systems. For the purposes of the ASSEMBLE system, the robot is modeled in terms of its basic capabilities such as grasp, free motion and certain fine-motion commands.

As stated earlier, the primary motivation behind the work on the development of an object model is the identification of the characteristics of objects that are essential for solving the assembly planning problem and its structuring. A hierarchical representation of objects has been developed, that emphasizes the surface characteristics and symmetry properties of objects, and relative locations of features on the object. The representation is built around the notion of features of objects that are particularly relevant for assembly, such as holes, threaded surfaces, etc. Precise shape information about an object is not required, instead shape and size information that precisely define the geometry of the features is required. The topological relationships between features are also explicitly represented in the object models. Features in turn are seen as comprising surfaces which come to be in specific relationships with surfaces of other features as a result of assembly operations. Thus, objects are viewed as collections of features, and features in turn are viewed as collections of surfaces. Information about objects is stratified into these three levels. Object information as required by this representation constitutes part of the input information to the ASSEMBLE program.

The assembly planning problem itself is viewed as a series of transformations into descriptions of the problem in a hierarchy of abstraction spaces. The description of the

problem at the object level is first refined to a feature level description. This refinement process is achieved through the use of geometric constraints associated with the operations, symmetry properties of objects, assembly heuristics and propagation of accessibility constraints. Feature level descriptions are refined into a set of spatial relationships between geometric entities associated with the features. These spatial relationships are derived by combining information regarding the shapes of the features with the essential kinematic relations suggested by the assembly operation. The intermediate representation of the problem as a set of spatial relationship goals allows ASSEMBLE to interface with a spatial reasoning system to identify whether the choices made in the course of problem refinement thus far are indeed viable. If so, the spatial reasoning system provides a way to combine the relationships to obtain tighter relationships. The spatial reasoning system used with ASSEMBLE is the RAPT system, developed at University of Edinburgh. RAPT refers to a language for describing assemblies by means of symbolic spatial relationships [45] as well as an associated inference engine that works with these symbolic spatial relationships and derives position expressions that represent the locations of objects [15]. The feature-level descriptions and spatial relationships are then to be used to generate the necessary grasp, free motion and manipulation commands.

The input to the ASSEMBLE system consists of a sequence of assembly instructions, each of which specifies an assembly operation and the objects the operation brings together. Information about the objects as prescribed by the representation also constitutes part of the input. The problem of determining the relative pose of objects with respect to each other, given incomplete information about the objects and an abstract specification of the task, is similar in its general goals to several problem-solving and planning systems in AI. In most AI planning systems the emphasis is on reasoning about actions and therefore the environment and the effects of actions on the environment are represented very simply, e.g., as predicate calculus formulae. In this work, the geometric information about objects and the geometric constraints imposed by assembly tasks are viewed as being critical in defining as well as reducing the search space.

The ASSEMBLE system comprises three subsystems: FIND-FEATURES, VERIFY and GENERATE-MOVE-COMMANDS. The FIND-FEATURES subsystem transforms an object-level description to feature-level descriptions; that is, corresponding to each

operation specified in the assembly task it generates hypotheses about what the mating features are. The VERIFY subsystem is responsible for validating the hypotheses generated by the FIND-FEATURES subsytem. As a part of the validation process, VERIFY transforms the feature-level description of an assembly task to the implied spatial relationships and interfaces with RAPT to verify that these spatial relationships can be satisfied simultaneously. The GENERATE-MOVE-COMMANDS subsystem uses the feature-level descriptions and spatial relationships derived by the VERIFY subsystem to generate the abstract robot-level commands that are the output of the ASSEMBLE system. Chapter 6 provides information on the details and and status of the implementation of ASSEMBLE. The possible sets of mating features leading to different interpretations of the input instructions are represented as nodes in a search tree and various sources of constraints are used to prune the search tree. This problem is treated as a planning problem in that the specified order of the assembly instructions provides a significant source of constraint in the choice of mating features. There is, however, an interesting difference between ASSEMBLE and other traditional planning systems. In most planning systems, the refinement of actions is specified, and the effects and preconditions of these actions are used to determine the order of these actions. In ASSEMBLE, the order of the actions is specified and the refinement of the actions is reasoned about.

The conceptual developments of the object model as well as the approach to decomposition and associated reasoning are concretized through the development of the program ASSEMBLE. The experimental assembly tasks run on ASSEMBLE serve to indicate the adequacy of the object models and the reasoning processes and to point out their limitations.

1.5 Organization of the Dissertation

The dissertation is organized into eight chapters, followed by a set of five appendices and a bibliography. This section presents a brief description of the contents of each of the following chapters.

Chapter 2 discusses related work in the areas of planning and robotics and relates the

research described in this dissertation to the literature.

Chapter 3 discusses issues pertaining to the development of a system architecture suitable for an intelligent robot system. Further, it provides a description of the specific architecture formulated, and adopted for this dissertation, and details the interface between the assembly planner and other subsytems. This architecture provides the framework for the research that is described in the dissertation.

Chapter 4 discusses issues relating to the representation of objects in the context of developing an assembly planner. The specific representations developed in the course of this research are described. Their potential and limitations are discussed in relation to other representations used for similar or related problems. Chapter 5 motivates and discusses the hierarchical decomposition of the assembly planning problem developed in this dissertation. The assembly planning problem is successively refined to levels of detail that correspond roughly to the levels of detail in the hierarchical object descriptions. The reasoning mechanisms that effect these refinements as well as the intermediate representations used are motivated and described.

Chapter 6 provides a description of the design and implementation of the ASSEMBLE system using the object models described in Chapter 4 and in accordance with the hierarchical decomposition structure described in Chapter 5. Details of the representations and algorithms used are provided and illustrated with examples. Delimitations, limitations and possible extensions of the system are discussed. Chapter 7 discusses the performance of the ASSEMBLE system on some experimental assembly tasks. The capabilities of the various components of ASSEMBLE are demonstrated, and possible extensions to the system are outlined. Appendices 7.1 through 7.5 provide details of input and output corresponding to the experiments discussed in Chapter 7.

Chapter 8 provides a summary, concluding remarks and outlines directions along which this research can be extended.

CHAPTER 2

REVIEW OF LITERATURE

The problem of planning an assembly as defined in Chapter 1 has not been addressed before. However, similar and related problems have appeared in the literature. This chapter attempts to summarize some of the salient efforts in these related areas. Several ideas arising from earlier work have influenced the design and development of the ASSEMBLE system. Assembly is made up of a sequence of actions, and the geometry of the objects in the assembly provide a significant source of constraints for the problem of planning an assembly. Therefore, the following two issues are central to the problem addressed in this work:

- representing and reasoning about actions, ordering actions, analyzing the interactions between them;
- representing the environment, i.e. objects, the robot and free space.

This chapter comprises two sections in which work that addresses each of these issues are discussed.

2.1 Actions: Representation and Reasoning

Representation and reasoning is central to much of the work in Artificial Intelligence. Specifically, the area of *planning* has addressed the issue of representing and reasoning about actions. Actions are seen as functions that effect state transformations, where the set of states, however large, is discrete. In the field of robotics, the actions are executed in continuous space, and consequently, the efforts in motion planning have taken an

analytical approach. This dissertation takes a state-operator oriented approach to the assembly planning problem, as do most AI planning systems. Thus, the various efforts in the AI planning literature are discussed in detail here. Some of the task planning work from the area of robotics are also discussed, to bring out the issues and constraints that arise from the requirement that the plan needs to be executed in a continuous domain.

2.1.1 Planning in AI

Breaking down a given, nontrivial task into actions that can be considered primitive has been a major concern of several problem-solving systems in AI. Specifically, research in the area of *planning* addresses the problem of arriving at a sequence of actions to achieve a specified goal. Several planning systems have been developed and the significant efforts among these are discussed in more detail in this section. This section attempts to describe how the research efforts in the planning area have developed, with many of the systems building on and extending the ideas developed in earlier systems. Many of the systems work with a state-space formulation of the problem. A few systems (MOLGEN, Wesson's system and Pengi) propose significantly different ideas which are of particular relevance in the domain of robotics.

Common to several of these systems is the way the planning problem is formulated as a problem in state-space. A *task* is specified by describing an initial state and a final state. A *goal* is a specification of a desired property and is equivalently used to refer to a set of states that satisfy the desired property. Thus, a task may also be specified by describing the initial state and a goal. An *action* produces a state transformation. Actions are characterized by their applicability conditions (or preconditions) and their effects. The problem is to arrive at a sequence of actions that transform the initial state to a final state. Typically, the approach adopted is one where the goal is reduced to a sequence of subgoals. The subgoals represent intermediate states, thus reducing the transformation from initial state to goal state to a series of transformations through these intermediate states. This problem decomposition is repeatedly applied until a sequence of subgoals is obtained such that each transformation required by the corresponding sequence of intermediate states is achievable by one of the specified repertoire of actions. Clearly, the crux of the problem, in this formulation, is in determining the *sequence* of

subgoals corresponding to a goal, that is, not only the *set* of subgoals but also the *order* in which they are to be achieved.

One of the earliest works in the area of planning is the General Problem Solver (GPS) [18]. GPS attempted to separate its general problem-solving methods from knowledge specific to the type of task at hand. The task dependent knowledge was encoded in objects (states) and operators for transforming the objects. GPS used a technique called means-ends analysis to address the subgoal selection process described above. First, it computed the differences between the initial state and the goal state. It then consulted a table that indexed the various operators according to their effects. The differences that would be reduced by these effects were counted and the operator that would reduce the least desirable differences was applied. Applying this operator would create a new state and this state would then be used as the initial state for the next iteration of the process. Here the ordering of the operators is a byproduct of the choice of operators resulting from the means-ends analysis approach.

STRIPS [20,21] is a robot planner that also viewed the problem in its state-operator representation. It used many of the ideas from GPS, specifically means-ends analysis. The STRIPS world consisted of a robot which could traverse several rooms with doors connecting them and move boxes from one room to another. A set of predicate calculus assertions was used to represent the world. Transformations effected by the operators were represented by add- and delete-lists associated with each operator. A resolution-based theorem prover was used to determine the operators to be applied (as opposed to the operator-difference table of GPS). STRIPS was also the first system to explicitly associate preconditions with operators. This allowed STRIPS to formulate subgoals that represented the satisfaction of these preconditions. One of the problems STRIPS had was the lack of organization in its space of operators and a consequent inability to discriminate between important and unimportant differences.

ABSTRIPS [50] is an extension of STRIPS, which included a hierarchy of abstraction spaces, so that operators that reduced critical differences were chosen earlier than ones that merely served as filling in the details. Defining the abstraction spaces was achieved through assignment of criticalities to preconditions of operators. Thus, when ABSTRIPS started planning, it planned to achieve only those preconditions that had the highest

criticality. If it succeeded in achieving this, it then attempted to achieve preconditions at the next level of criticality and so on, filling in the details between the steps of the plan from the previous level. However, ABSTRIPS was not capable of backtracking between hierarchical levels. Thus, the major contribution of ABSTRIPS is the introduction of hierarchical planning.

All of the planning systems described above emphasized the subgoal selection process rather than their ordering. The assumption that the subgoals of a problem are independently achievable and that the order in which they are achieved makes no difference to the outcome is termed the *linearity assumption*. However, interaction between subgoals was recognized to be a significant problem even in simple and well-constrained domains such as the "blocks world." This recognition led to the development of a number of systems that emphasized the ordering of subgoals. The significant systems among these are described below.

HACKER [55] is a system that addresses the issue of interacting subgoals. HACKER solved problems in the blocks world by making the *linearity assumption*. The system created an initial plan based on this assumption. At that point a set of procedures called *critics* was invoked by HACKER to identify and repair problems in plans caused by interactions. If a conflict was detected, the system consulted a library of patches to see if there was a way to resolve the conflict. HACKER was often able to repair the plan by changing the order in which subgoals were satisfied. Once such a patch had been made, a version of the situation leading to the patch and information about the fix used were put into a library of critics. This library was consulted if a similar situation came along so that the same fix could be used if applicable. A major disadvantage of HACKER was that it sometimes did a lot of wasted work. The system would usually produce a correct plan, but only after first building a defective plan and then debugging it to form a new plan. However, there are some simple problems in the blocks world for which HACKER could not produce an optimal plan.

INTERPLAN [56] was a planning program for the blocks world developed at the University of Edinburgh. The system worked by performing a search to find the correct linear ordering of the subplans. It was able to reorder existing plans and to backtrack when contradictions were recognized. It accomplished this by using a Table of Multiple

Effects which represented interactions between subgoals that would be used conjunctively. INTERPLAN abstracted *holding periods* from the original goals, i.e., periods during which the goals were assumed to be true. It then analyzed them in order to plan how to move goals to ease conflicts. By analyzing conflicts before backtracking, INTERPLAN could sometimes outperform HACKER.

Waldinger [62] developed another approach to the problem of interacting subgoals and applied this to problems in the block world as well as in program synthesis. The approach was called *goal regression*. It involved creating a plan to solve one of several subgoals followed by constructive modifications to achieve other subgoals. It differed from HACKER in that it used notation about protection of goals to guide the linear placement of actions in the plan. Thus, rather than building incorrect plans and debugging them, it built partial sequences of actions without imposing an order on the sequences at first. The term goal regression is suggestive of the way the program worked, moving goals backward through a sequence of actions to a point in the sequence at which they did not interfere.

NOAH [51] is the first planning system that introduced the notion of nonlinear plans (it did away with the *linearity assumption*) with a *least commitment* approach, in which any ordering decisions were deferred until there was a reason to commit to such a decision. It retained the hierarchical planning of ABSTRIPS but opted for a more procedural approach to planning. For each level of abstraction in NOAH, operators were created which indicated the steps in the next lower level of abstraction as generalized procedure calls. The order of the procedures at the lower level of abstraction was not anticipated, but was left for NOAH to determine. NOAH represented plans as procedural networks at various levels of abstraction. NOAH started out with a specified goal and determined an abstract plan to achieve this goal by invoking the appropriate procedure. The plan was then successively refined until primitive actions were obtained. Central to NOAH is its method for determining the order in which conjunctive subgoals are to be achieved. At each level of plan development, NOAH employed *critics* that analyzed the current plan for interacting subgoals. The system also used a Table of Multiple Effects (similar to INTERPLAN), a global data structure, which would make the critics more efficient. NOAH imposed no ordering on the subactions of the plan until the critics indicated

the need for such an ordering. If an action for one goal deleted a precondition for a conjunctive goal, then the action with this precondition was moved forward. Only after this criticism phase was the plan expanded to the next lower level. In NOAH, interactions were examined on every level, and it was only possible to be certain of freedom from interaction when the atomic level was reached. When NOAH placed actions in a plan, it did so in one particular way and could not change its decision later. Also, it could not deal with interferences between actions that cannot be fixed by reordering the actions.

Tate [57] developed a system called NONLIN which extended NOAH to facilitate backtracking. NONLIN generated and maintained for future use alternative operator choices, alternative instantiations and ordering of operators. These alternatives were kept at explicit *choice points* in the plan and could be used when an initial plan failed or when it was useful to generate more than one solution. NONLIN maintained the *goal structure* of a plan, independently of the chronological links in the plan, which would contain information about the purpose behind the introduction of specific actions, and the interrelations between goals. The explicit provision of the *goal structure* facilitated the detection and correction of interacting subgoals. NONLIN also distinguished between effects of an action irrelevant to the application of later operators and the interactive effects.

SIPE [65] is essentially an extension of NOAH which adopts the basic representation mechanism, namely, procedural nets and the use of critics for analyzing plans for subgoal interaction. The extensions SIPE provided were in terms of explicit representations for conceptual entities relevant for the planning process. It provided a way to represent resources and reason about them. It also allowed for the representation of the rationale behind each action. In addition, the system provided a language in which constraints on object selection could be represented. It also provided a mechanism by which side-effects of actions may be deduced, as opposed to the intended effects which are explicitly stated in the representation of actions.

The systems described above adopt an iterative stepwise refinement approach to plan development, where complete plans are developed at each level of the hierarchy before plan refinement at a more detailed level is attempted. In contrast, the systems we discuss below adopt an opportunistic style of problem solving, making decisions about portions

of the plan where adequate information exists and using these to guide the decisions about other portions of the plan.

Hayes-Roth and Hayes-Roth [22] developed a model of planning with a view to emulating human planning behavior. The model adopts several of the basic features of Hearsay-II [17]. Specifically, it employs an opportunistic problem-solving approach, viewing the planning process as the cooperative effort of several knowledge sources, which apply their specific expertise to different aspects of the problem. These knowledge sources develop islands of confidence that represent subproblem solutions or decisions, which direct the search for an appropriate solution to the problem. The model allows communication between the knowledge sources via a blackboard and includes an intelligent scheduler to control knowledge source activity.

MOLGEN [53,54] is a planning system developed for designing experiments in molecular genetics. MOLGEN, like NOAH, is a hierarchical planner and solves the given problem in levels of increasing detail. However, MOLGEN maintains an operator hierarchy as well as an object hierarchy. Thus, abstraction is introduced in the objects manipulated by the problem, in this case, laboratory objects. The primary tool for problem solving in MOLGEN is the use of constraints. Constraints allow the system to refine objects and operators. Explicit constraints tie together interacting subproblems. Refinement follows a formulate-propagate-satisfy cycle, whereby the order in which abstractions are refined depends on the way constraints are propagated in a particular plan, rather than on a fixed criticality ordering. Another important contribution of MOLGEN [54] is the notion of meta-planning, where the decision-making knowledge is organized in a layered control structure which separates decisions about the problem domain from decisions about the planning process. At the highest level, the system chose to focus on a subproblem and refine it further, suspend it when no information was readily available for further refinement, and resume it when more information became available. Such information became available through the propagation of constraints, facilitating an opportunistic style of problem solving in MOLGEN. MOLGEN introduces some interesting approaches to dealing with subgoal interaction, but uses constraints primarily for object selection and does not include sophisticated representation mechanisms for processes.

Wesson [64] built a planning system for the world of the air traffic controller. The

system distinguishes between the *real world* and the idealized *problem solver's world.* The problem solver's world is updated periodically with real world data. The system operates by simulating, in its idealized world, the movements of aircraft in an air traffic controller's environment. Using the simulator to look ahead and predict potential conflicts, the program attempts to resolve these conflicts by changing current flight plans and checks the revised plans for validity by rerunning the simulator. The system significantly differs from other planning systems in that it accommodates changes in the context in which the problem solving occurs by periodically updating its view of the world, while other systems assume that the only changes to the world model occur as the effects of the problem solver's actions.

Recent work by Agre and Chapman [1] attempts to build a model of a theory of activity in the context of situations that are complex, uncertain and immediate. Their stance is that action does not derive from the execution of plans and the corresponding framework of problem solving and reasoning with representations. They posit that activity mostly derives from very simple sorts of machinery interacting with the immediate situation. They are investigating their ideas by developing a program, Pengi, that plays a commercial arcade video game called Pengo. The system uses the notion of *indexical-functional aspects* whereby only objects that are relevant to the problem solver's current goals are represented as opposed to explicit representation of all objects in the problem solver's environment. Actions are represented as *routines* which represent patterns of interaction between an agent and its world. Assessment of current situations is carried out by *visual routines*. The basic premise is that the problem solver has several goals and the choice of actions is made opportunistically based on an evaluation of the current situation. The implementation is in progress and no results are available at this point.

All of the systems discussed thus far focused on the refinement, selection and ordering of operators. The problem description, namely, the specification of initial and final states, is assumed to be clearly stated. MOLGEN is the only planning system that provides an approach to operator refinement as well as problem refinement with the introduction of the notion of abstract objects. The problem of assembly planning is one where the initial description of the problem does not include all of the relevant details for executing the actions implied in the description. The system attempts to fill in the details to arrive at

a precise definition of the final state.

2.1.2 Task Planning

The planning efforts in the field of robotics itself have centered around the development of task-level programming languages that would require the user to specify a task in terms of the physical relationships between objects rather than the robot motions required to carry out the task. A peg-in-hole insertion task, for example, would be described as "Insert Peg in Hole" instead of the sequence of robot motions needed to accomplish the insertion. A detailed analysis of the problem can be found in [33].

While it is tempting to cast the task planning problem in terms of the formalisms adopted by AI planning and problem-solving systems, there are some key differences which need to be taken into account.

The specified goal of a task planning system needs to be broken down into a constituent set of motion primitives. We will assume here that grasping, gross motion and fine motion abstractly characterize these motion primitives. Typically, the order in which these primitives need to be carried out to accomplish a task is not subject to variation. For example, the "Insert Peg in Hole" task mentioned above can be carried out by executing (a) a gross motion to the vicinity of the peg; (b) grasp of the peg; (c) gross motion to the vicinity of the hole; (d) a guarded motion until the peg makes contact with the hole; (e) a fine motion that accomplishes successful insertion; and (optionally) (f) release of the peg (if this was not already a part of the fine-motion strategy). The challenge, however, is in the choice of parameters that can clearly define these motions. The parameter spaces are often continuous, and feasible regions are constrained by the geometry of the environment and the robot. Special purpose planners for each of these motion primitives are subjects of active research. These subproblems are also highly interdependent. For example, the choice of grasp site on an object determines what motions of the hand are required to position the object. The choice of grasp site, in turn, is determined by subsequent operations; for example, a finger cannot be against a surface that is to be against another surface. Similarly, the existence of a path to a destination might depend on how the object is held in the hand. Additionally, each operation makes

certain assumptions about accuracy in location and shape which affect subsequent operations. It is clear that there has to be a good deal of communication between these special purpose planners, and choices made by one subsystem should serve to constrain the possible choices for other subsystems. An integrated task planning system using skeletal plans and constraint propagation to iteratively select parameters was proposed in [35] but has not been implemented. It views the task planning problem as comprising the following subproblems: parts feeding, layout, fixturing, fine motion, grasping, and gross motion. It further illustrates how the subproblems interact and proposes the propagation of symbolic algebraic constraints as a possible means of coping with the interactions.

A second significant difference is in the accuracy of the world model. Clearly, planning motions of the robot requires geometric models of the environment; the legal motions of an object are constrained by the presence of other objects in the environment and the form of the constraints depend on the shapes of the objects. Even in the case of assembly planning, which does not involve planning the motions of a robot, the richest source of constraints comes from the geometry of the objects and task. Consequently, the representation of the environment, or world model, contains information that is primarily geometric and geometric reasoning is a significant problem-solving component.

While none of the task-level programming languages proposed have been completely implemented, we discuss below the significant efforts, their original intent and their accomplishments.

The LAMA system was designed at MIT [30] as a task-level language, but only partially implemented. The system viewed the translation process as consisting of two phases: first, the assembly description, with many missing details, was to be converted into an assembly plan; and second, the assembly plan was to be converted into a manipulator program. The implementations focused on the second phase, specifically on the subproblems of grasping, tailoring assembly strategies and error prediction.

AUTOPASS, at IBM [28], defined the syntax and semantics of a task-level language and an approach to its implementation. A subset of the most general operation, the PLACE statement, was implemented. The major part of the implementation effort focused on a method for planning collision-free paths for Cartesian robots among polyhedral

obstacles.

RAPT [46] is an implemented system for transforming symbolic specifications of geometric goals in terms of spatial relationships, together with a program which specifies the directions of the motions, but not their length, into a sequence of end-effector positions. RAPT's emphasis has primarily been on task specification.

2.2 Representation of the Environment

The analysis of interactions between the actions is cast in a state-operator framework in the assembly planning system detailed in this dissertation. Much of the problem refinement which is a larger component of the assembly planning problem is based on geometric reasoning. The reasoning relies on the representations of objects allowed by the object models. Chapter 4, which describes the object models, provides an account of related work that focuses on representations of objects and their assembly. This section will discuss some of the work in AI that has laid a stronger emphasis on environmental representations for problem solving and a brief sketch of solid modeling systems and their potential for robotics.

Of the early AI planning systems, Fahlman's work on construction planning in the blocks world domain [19] is the only problem-solving system that uses a detailed representation of objects. Each object in the environment is represented by a complete 3-dimensional model indicating the size, shape, position and orientation of each block. Although the shapes considered are restricted to rectangular blocks and right-angled wedges, the geometry and physics of these blocks play a primary role in the problem solving system. Another notable system is Brooks' ACRONYM [7] which is a model-based visual recognition system. The system uses a generalized cone representation for describing objects, and relies strongly on the geometric and spatial relationship constraints derived from these models to guide the recognition process.

Almost all of the systems and proposed approaches in the motion planning domain, which includes path planning, grasp planning and fine-motion planning, work with polyhedral approximations of the environment.

Work in CAD/CAM has given rise to several solid modeling systems. The term *solid modeling* encompasses a body of theory, techniques, and systems focused on informationally complete representation of solids, that is, representations that permit any well-defined geometrical property of any represented solid to be calculated automatically. The CSG representation and Boundary Representation are used predominantly in solid modeling systems. A good overview of research in this area can be found in [49].

While solid modeling systems provide informationally complete representations, robotic applications generate some specific needs. Robotics is typically about collections of objects related to each other in specific ways, and not about individual objects. Ambler [2] discusses the requirements robotic applications place on solid modeling systems. While some of these relate to motion planning, the requirements that are of relevance to the assembly planning problem are listed below:

1. Easy access to surface descriptions;
2. Ability to specify surface features which do not contribute to the shape of the body concerned;
3. Establishment of a reference system for surface features which is stable under spatial transformations;
4. Ability to check the topological similarity of parts descriptions;
5. Ability to do robust and efficient interference detection over a set of bodies some of which may be in contact;
6. Ability to detect symmetries of features;
7. Representation of tolerances on dimensions of parts in a way which allows us to reason about the buildup of tolerances over the whole assembly.

The assembly planning problem requires a characterization of the surfaces as well as the volume of objects. The objects models developed in this work attempt to identify the key characteristics of objects that are of relevance to the assembly planning problem, without making any assumptions about reliance on informationally complete representations such as a solid modeler would provide.

2.3 Summary

This chapter has presented some of the existing work in the areas of planning and problem-solving in AI, task planning, and some of the existing approaches to representing objects and the environment in which a problem solver operates.

With respect to the issue of representing actions, the work described in this dissertation adopts a state-operator formalism as do traditional planning systems. The representation of plans in a network of goals and subgoals is similar to the representation introduced by NOAH and extended by NONLIN. The approach adopted by the assembly planning system for dealing with subgoal interaction is one of *goal regression*, similar to Waldinger's system [62], where the analysis of actions proceeds in reverse of the order in which actions are executed. The hierarchical representations of objects and goals and their successive refinement is similar to the object and operator hierarchy of MOLGEN. There basic ideas in the existing literature have been used and adapted to suit aspects specific to the assembly planning problem.

In contrast, this work departs significantly from existing work in the area of object representation. The represenation developed and the related problem decomposition approach are the major components of this dissertation. They are presented and discussed in detail in the remainder of this dissertation, specifically, chapters 4 through 7.

CHAPTER 3

A SYSTEM ARCHITECTURE SUITABLE FOR INTELLIGENT ROBOTICS

As mentioned in Chapter 1, one of the goals of this research is to develop an assembly planning system that can be realistically integrated with other necessary components of an intelligent robot system. This chapter describes and motivates *dynamic planning*, an approach to planning suitable for the robotic domain. It then outlines a system architecture identifying the necessary components of a robot system and how these components relate to each other. Both the dynamic planning paradigm as well as the system architecture proposed are conceptual developments. The actualizations of these is beyond the scope of this dissertation. However, they provide the framework within which the assembly planner, the development component of this dissertation, is couched. The interface between the assembly planner and various components of the system to which it relates are detailed in this chapter.

3.1 Dynamic Planning

The planning problem in the robotic domain can be characterized as one of transforming a set of task specifications provided by the user, which are at the level of identifying what objects are involved in the task, to a set of commands executable by a robot control system. The approach adopted by traditional planning systems [20,51,65] is based on the specification of an initial world model and the changes effected by each primitive action on the current state of the world. However, the characterization of the initial world and the effects of actions are rigidly specified. In the case of a robot system operating in the assembly domain, the information available to the system is often inaccurate. There are several sources of uncertainty: uncertainty in the geometric models of objects in the

robot's environment; uncertainty in the information obtained from sensors; and uncertainty arising from the inaccuracies of the manipulator control system. The subactions, such as grasp, move and manipulate are parameterized and the choice of parameters for each of these subactions is highly interdependent. Thus, all the information pertaining to the applicability of operators may not be known *a priori* and it is difficult to predict the effects of actions with certainty. In an effort to contend with dynamic changes in the environment and inherent uncertainty, the role of planning is extended to include monitoring for the successful execution of the plan, and necessary replanning as well. In essence, the planner does not function as an off-line planner, but instead deploys the sensors to monitor the environment and correspondingly effects changes in the plan, where necessary. Such an approach to planning is referred to as *dynamic planning* [61].

The salient characteristics of a dynamic planning component of a robot system are the following:

1. It *incrementally* constructs a plan, starting with *inadequate* information and procuring information and *refining* the plan as it proceeds.
2. It *dynamically revises* its plans and goals corresponding to perceived changes in the environment. In particular, the plan is considered to need revision when the environment resulting from the execution of the plan fails to meet certain *specified criteria.*
3. Planning and execution may be *interleaved*; it is not necessary that all the planning precede all the execution.

3.2 Proposed System Architecture

One of the primary goals of robotics is the mechanization of intelligent behavior. The structuring of an intelligent robot system should, therefore, reflect some of the structural features that are present in human behavior. In this regard, the incorporation of feedback into the system and hierarchical structuring are considered to be two key features that need to be retained in the design of a robot system.

The overall system design consists of the strategic planner, the tactical planner, object database, sensory systems, controllers for the arm, wrist and hand and the necessary

hardware. The strategic and tactical planners, control systems as well as the manipulator hardware constitute the control hierarchy while the sensors and the preprocessing systems constitute the sensory hierarchy. The object database serves as a repository of information about the environment, updated by the sensory preprocessors and accessed by the units in the control hierarchy. A schematic of the overall system and the relationship between the various subsystems is shown in Figure 1.

The assumptions regarding the basic characteristics of the underlying hardware are outlined below:

- a six degree of freedom robot arm which has three positional degrees of freedom, and a three rotational degrees of freedom wrist;
- a dextrous hand mounted on the wrist;
- availability of force information from the wrist and from strain gauges mounted on the fingers of the dextrous hand;
- tactile sensors mounted on the fingers and palm of the hand;
- a visual system that produces location information (position and orientation in 3-dimensional space), and also some visual features. This would involve a stereo camera system, or a camera and range-finder, connected to a preprocessor system.

Above and separate from the sensor hardware are some preprocessing elements. A tactile preprocessing system implements useful tactile algorithms as has been described in [5] and [43]. Tactile and force information will be dealt with in a unified fashion. The visual preprocessor takes the input from the camera system and produces object features. The preprocessor could use information from the object database, allowing it to classify objects.

The object database contains information about the various objects in the environment including the robot itself. These object models integrate standard structural information about an object along with functional information as to how the object may be recognized, inspected, and manipulated. The planning systems use these models of objects for reasoning about the specific problem they are concerned with. When features relevant to the planning process are absent in the model, the object database module would be responsible for directing the sensory systems to attempt to obtain this information. In addition, the knowledge about objects embodied in the object database could

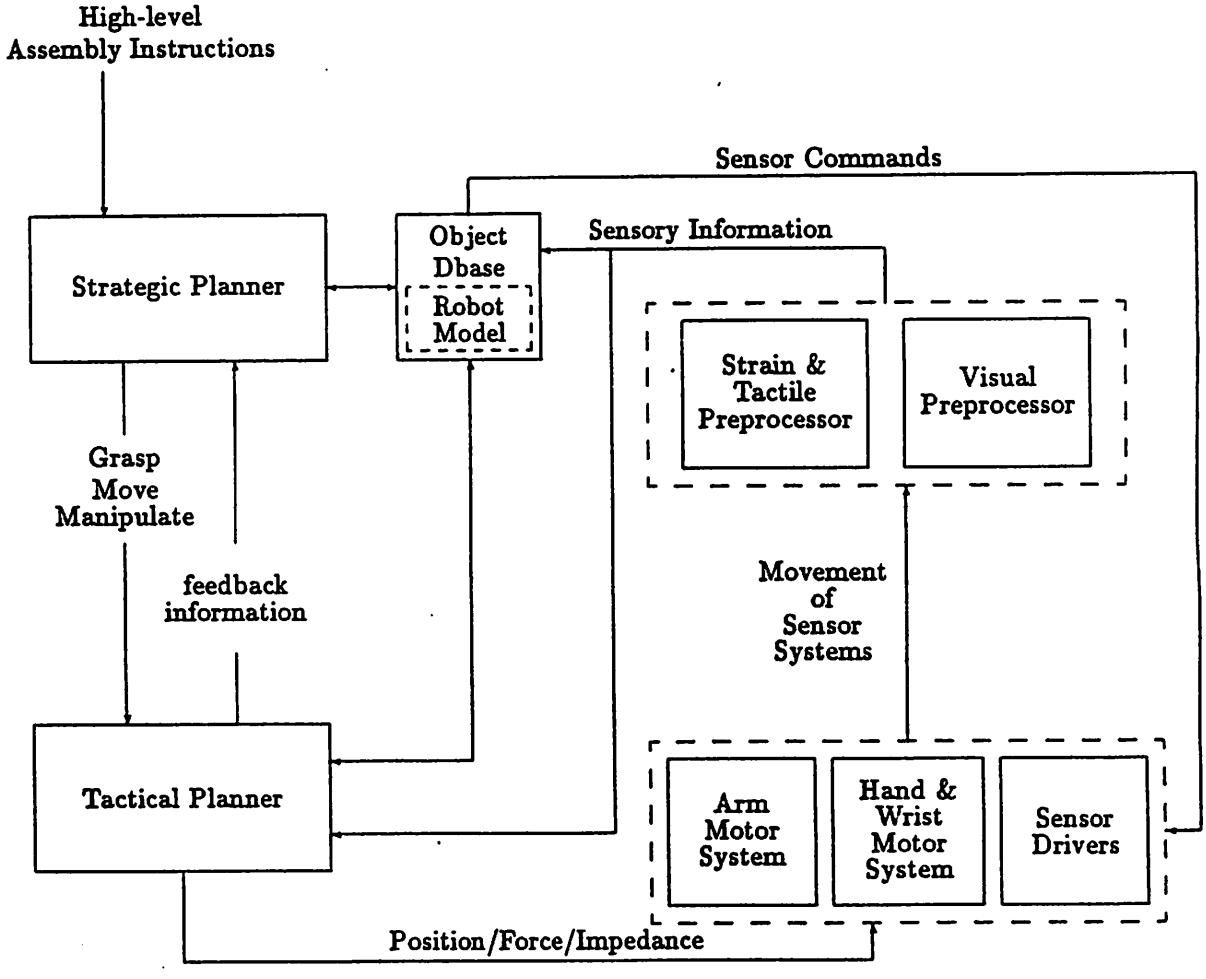


Figure 1: Schematic of the Overall System

be used as constraints to guide the identification of objects that are of interest to the current planning problem. It should be noted that the ability of the sensory preprocessors to provide information at the level of abstraction required by higher levels of planning is currently very limited.

3.2.1 The Control Hierarchy

The control of the robot system at the lowest level is separated into a hand controller, an arm controller, and sensor drivers which effect movement of sensory systems such as the camera. Separate controllers are used for arm and hand, because their respective functions are quite different. The arm is used only for gross position placement, fine tuning being accomplished by the fingers of the hand. The arm controller allows the hand to be directed along a given trajectory. The hand controller allows the fingers of the hand to be placed in particular configurations, and also allows the stiffness of the fingers to be controlled.

The high-level robot control is divided into two hierarchical units: the strategic planner, which takes a broad definition of the assembly task and, using the object database, generates a more specific sequence of parameterized operations; and the tactical planner, whose input is a restricted set of task-level operations, and whose output is a series of position and stiffness signals to the lower-level arm and hand controllers. The following sections illustrate the distinction between strategy and tactics, and describe the structure of the strategic planner in more detail.

Feedback in the system is modeled both through the use of sensory information at various levels of the control hierarchy as well as through information fed back from lower level control units to levels above.

3.2.2 Strategy and Tactics

The problem of decomposing and successfully executing a specified assembly task requires decision making at several levels. Primarily, the problem is decomposed into one of developing the *strategy* required to accomplish a task and the *tactics* required to fuse

the units of action suggested by the strategy to successfully execute the task [39]. Thus, in planning at the strategic level, the system makes discrete choices about the actions that are appropriate, their ordering and the selection of parameters that define the actions more precisely. Planning at the tactical level involves fine tuning of the parameters so that the intended behavior is accomplished as well as the orchestration of the units of action to achieve the desired effect. To illustrate the distinction between strategy and tactics, a pick-and-place task is discussed. The task of moving an object from position A to position B may be broken up into a sequence of Move to A, Grasp object, Move to B, Place object, and Release. Detailed strategies for Move to A and Move to B may be developed so that piece-wise linear paths are obtained. Similarly, grasping the object requires selection of grasp site and grasp configuration. All of this constitutes the strategy for accomplishing the task. The tactics required to execute this task involves ensuring stability of the grasp and making minor variations in grasp site and configuration to maintain force equilibrium, making the transition from the free motion that will take the object to the vicinity of location B and the guarded motion that will accomplish the placing, or making local modifications to the path to maneuver the object in the vicinity of an obstacle, etc. In essence, the tactical planner oversees the execution of the actions specified by the strategic planner, fine tuning the parameters that qualify these actions in order to successfully accomplish the goals of these actions.

The strategic planner, in order to select, sequence and parameterize the necessary set of actions, requires a model of the environment, including the specific robot device that is to carry out the task. In addition, it requires models of the tasks and reasoning mechanisms to aid in the selection, sequencing and parametrization of the actions. Where information about the environment is inadequate, the strategic planner may employ a *strategy* of invoking sensors to obtain the information or using the reasoning mechanisms to derive such information. The tactical planner, however, works with knowledge of a localized segment of the environment; the environmental knowledge is not as abstract. Further, the tactical planner relies strongly on feedback through sensors for the monitoring of execution of these actions and the appropriateness of the parameters currently in use.

3.3 The Strategic Planner

For the domain of assembly tasks, developing the strategies required for robotic execution of these tasks involves reasoning about constraints arising from two distinct sources:

1. constraints arising from task and object characteristics;

2. constraints arising from the specific configuration of the robot being used.

Thus, the strategic planner is further decomposed into components that allow the separation of the two considerations listed above. Constraints arising purely from task and object characteristics are dealt with by an assembly planner which determines the relative pose of the objects involved in the assembly and generates a sequence of grasp, free motion and manipulation commands. These are then passed onto a grasp planner, path planner, and fine-motion planner which develop detailed strategies for the specified motions (see Figure 2).

Development of strategies for free motion is handled by a path planner which works with the free motion commands generated by the Assembly Planner, a model of the robot arm and its kinematics, models of the objects in the environment, and generates a piecewise linear path that is safe from collision. This path is then passed on to the tactical planner which interacts with the controller for the arm and monitors the traversal of the path.

Similarly, a grasp planner works with the object and operation information provided by the assembly planner, the detailed geometry of the object and the characteristics of the manipulator being used to determine the grasp site as well as the grasp configuration. The detailed strategy generated by the grasp planner can then be executed by the tactical planner which interacts with the necessary controllers.

A fine motion planner receives as input a specification of the operation, and a specification of the axis along which motion is desired. Choosing a strategy for fine-motion requires analysis of local geometry, physical characteristics such as friction between the objects, and would depend on the repertoire of control strategies that is available for use.

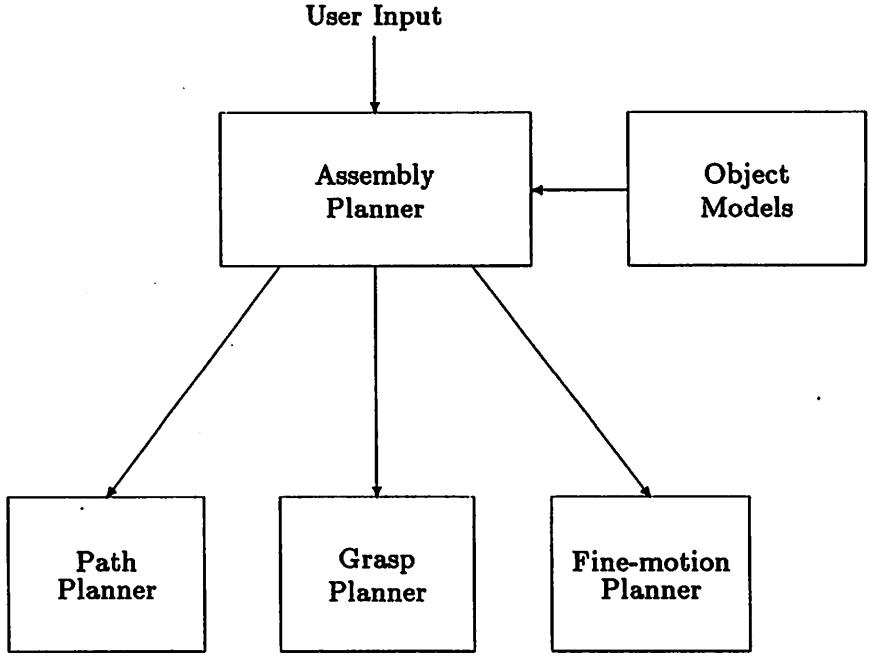


Figure 2: Structure of the Strategic Planner

This dissertation elaborates on the assembly planner which functions at the strategic level. The following describes the interface between the assembly planner and other subsytems as well as its interface with the user.

3.3.1 Interface between the Assembly Planner and other Subsystems

The problem domain that this dissertation addresses is that of simple assembly tasks. The parts sit in the workspace of a robot and the planner is given a very high-level plan for the assembly of the finished product. The high-level instructions will refer to objects by name and will not generally include any specification of parameters for actual robot movements, such as position, velocity, stiffness, etc. These instructions are then broken down into more detailed instructions using knowledge present in the system about assembly operations and about object information present in the object database and updated using sensors. The assembly plan is thus refined at several levels of detail until the plan can be expressed as a set of commands that may be directed at the grasp planner, path planner and fine-motion planner.

3.3.1.1 User Interface

The problem that the assembly planner is faced with is one of starting from a set of initial instructions which are of the form:

operation *object1 .. object2*

Some examples are:

Fit X into Y
Thread U with V
Place A on B

Starting with a set of instructions of this nature, and a description of the objects present in the workspace available from the object database, the assembly planner attempts to break down the given instructions into a series of output commands to the grasp planner, path planner and fine-motion planner. In this dissertation, the following operations will be considered: INSERT, THREAD, DOUBLE-INSERT and PLACE. A

study reported in [42] shows that insertion and screw operations are among the most common in typical assembly tasks and account for over 70% of assembly tasks in the sample set considered for the study.

3.3.1.2 Interface with the Path planner

As noted in Figure 2, the assembly planner sends commands specifying object or manipulator destinations to the path planner. In the following, the current working set of free motion commands that the assembly planner communicates to the path planner are listed.

Free Motion Commands:

All of the following commands assume that the robot hand is currently holding an object. They are expressed in terms of a relation that is required to exist between a point, line or plane in the object coordinate frame and a corresponding entity in the workspace coordinates. Axes and planes are oriented; each axis has a specified direction (so that two collinear axes may be aligned or opposed), and each plane has a specified normal. The same set of commands is applicable to the situation where the robot hand is free as well. In this case each of the commands is interpreted as specifying a desired relationship between the hand coordinate frame and workspace coordinates. These free motion commands are sent to the path planner and the trajectory produced by the path planner is then passed on to the tactical planner.

oppose(*axis1, axis2*): *axis1* is specified in terms of the coordinate frame of the object being held, while *axis2* is specified in terms of the workspace coordinates. The command instructs the robot to reorient the object in such a manner that the two specified axes are collinear and opposed in direction.

align(*axis1, axis2*): *axis1* and *axis2* are to be collinear with their directions being the same.

parallel(*axis1, axis2*): *axis1* and *axis2* are to be made parallel with their directions being the same.

xparallel(*axis1, axis2*): *axis1* and *axis2* are to be parallel with their directions being opposite.

place(*point1, point2*): This command requires the robot hand to move the object being held such that *point1* specified in terms of the object coordinate frame coincides with *point2* specified in the workspace coordinate frame.

on(*point1, line*): This command requires that *point1*, which is specified in the object coordinate frame, lie on the specified line. The parameter *line* is specified in the workspace coordinate frame.

on(*axis1, plane*): *axis1* is to be made to lie on the specified plane.

The path planner, when successful in finding a collision-free path to achieve a pose satisfying the desired relation, passes this path to the tactical planner. When it fails to find a path, failure indication is sent back to the assembly planner along with the identification of objects in the workspace that act as obstacles, for possible use in plan modification. There is still a possibility that the tactical planner may succeed or fail in accomplishing the pose. The feedback from the tactical planner on all of these commands is an indication as to whether the operation was successfully executed, the part was dropped, or if there was a failure to achieve the required pose. In the case of failing to achieve the specified position and orientation while still holding the object, the tactical planner passes back to the assembly planner the current pose of the object and the path segment that is left unexecuted.

3.3.1.3 Interface with the Grasp Planner

As detailed in [4] the grasp action is essentially carried out in two phases: the preshape phase, where the hand is configured to the preshape posture and carried to the vicinity of the object, and the grip phase where the hand acquires the object in a manner suitable for subsequent manipulation.

Grasp(*object, operation*): The command instructs the tactical planner to grasp the specified object by providing a set of grasp parameters along with the object identification. A more detailed description of the command parameters is given below:

object: The object parameter specifies the mating feature and its normal axis. The normal axis of a feature specifies the direction of the outward pointing normal to the surface of the feature. This provides a constraint to the grasp planner in that the chosen grasp site and type of grasp should allow for a motion degree of freedom along this axis.

operation: This parameter specifies the assembly operation to be executed in order that the grasping subsystem may choose an appropriate grasp site and configuration. For example, the grasp site and configuration selected to pick up and *place* a bolt at a specified location will be different from the site and configuration chosen if the manipulator is required to *insert* the bolt into a hole and *screw* it on.

The feedback from the tactical planner to the grasp planner is a signal indicating whether the Grasp command was successfully executed. In case of failure, it would indicate whether the failure occurred in the preshape or grip phase of the operation. A failure in the preshape phase would indicate that due to the presence of other objects in the environment the grasp configuration chosen, and consequently the preshape configuration, is inappropriate. A failure in the grip phase may arise either from a failure to acquire the object stably, or because the resulting grasp configuration overconstrains the object for the subsequent manipulation.

3.3.1.4 Interface with the Fine-Motion Planner

These commands assume the object is being held by the hand.

insert(*insertion-axis, insertion-amount*): Requests the fine-motion planner to perform an insert operation which involves a translation motion by the specified amount along the *insertion-axis.*

thread(*rotation-axis, rotation-amount*): Requires the fine-motion planner to execute a thread operation which in turn translates into a rotational motion by the specified *rotation-amount* about the *rotation-axis.*

double-insert(*insertion-axes, insertion-amount*): Requires the fine-motion planner to execute a double-insert operation which translates into a translation motion by the specified *insertion-amount* along the *insertion axes*.

place(*axis*): The place operation is essentially a guarded move which is a translation along the specified axis until contact is detected.

The feedback from the tactical planner on any of the manipulation commands is an indication of whether the execution of the operation succeeded or failed. In case of failure of insert or double-insert operations, the tactical planner returns the actual insertion amount along with the failure flag. In the case of failure of thread operations, the tactical planner returns the amount of rotation achieved, translation of the object along the axis of rotation and the termination torque.

3.4 Summary

This chapter has presented a system architecture suitable for intelligent robotics. The design presented emphasizes the need for hierarchical structuring and use of feedback in the development of such systems. The development component of this dissertation focusses on one component of the overall system, namely, the assembly planner. The interface between the assembly planner and other components of the system are detailed. The design, implementation, and evaluation of the assembly planner constitutes the remainder of this dissertation.

CHAPTER 4

OBJECT MODELS

4.1 Introduction

Planning an assembly task requires the solution to several subproblems: planning the relative pose of objects, planning a grasp, planning a collision-free path, etc. However, each of these problem-solving units focuses on different environmental features. The organization of the information relevant to a component should facilitate the solution of the subproblem it addresses. The object database of Figure 1, accordingly, should support these different representations while maintaining consistency. This chapter describes an abstract model of objects that has been developed for the assembly planning problem as defined in Chapter 1. The object representation used by the ASSEMBLE system is based on this model. The motivation behind the development of this model is the identification of object characteristics that play a key role in the solution of the assembly planning problem and the structuring of such information. No assumption is made as to the source of this information; that is, the information may either be derived from a CAD database or obtained through sensors.

The model developed is suitable for the class of rigid objects. The information about each object is hierarchically structured as object attributes, information pertaining to assembly features and the decomposition of these assembly features into constituent surfaces. The model emphasizes the characteristics of surface features of objects that are useful for assembly, the symmetry properties of objects, and the relative locations of these features on the object.

Some general information about an object and its gross dimensions is associated with an object. More detailed information about an object is associated with entities that

denote the features of an object. A *feature* is defined as "a specific geometric configuration formed on the surface, edge, or corner of a workpiece intended to modify or to aid in achieving a given function" [13]. Such structuring of object information has been motivated by the fact that in most assembly operations, it is the specific features of objects that dictate how these objects may be assembled together. The features considered are similar to the ones considered by Popplestone, Ambler and Bellos [46] in describing spatial relationships between features.

4.2 Object Representation

Each object in the assembly workspace is modeled by specifying a set of attributes which are organized hierarchically as a tree as shown in Figure 3. The first level of the hierarchy includes information pertaining to the object as a whole. These units of information are termed *object-attributes*. Subsequent levels of the hierarchy contain units of information relating to the features which are referred to as *feature-attributes*. The object-attributes that are included in the model are:

- shape,
- size,
- volume,
- surface area,
- weight,
- α-symmetry,
- β-symmetry,
- position,
- assembly features.

The size of an object is specified as its length, width, and height which are derived as the dimensions of the smallest rectangular parallelepiped into which the object will fit.

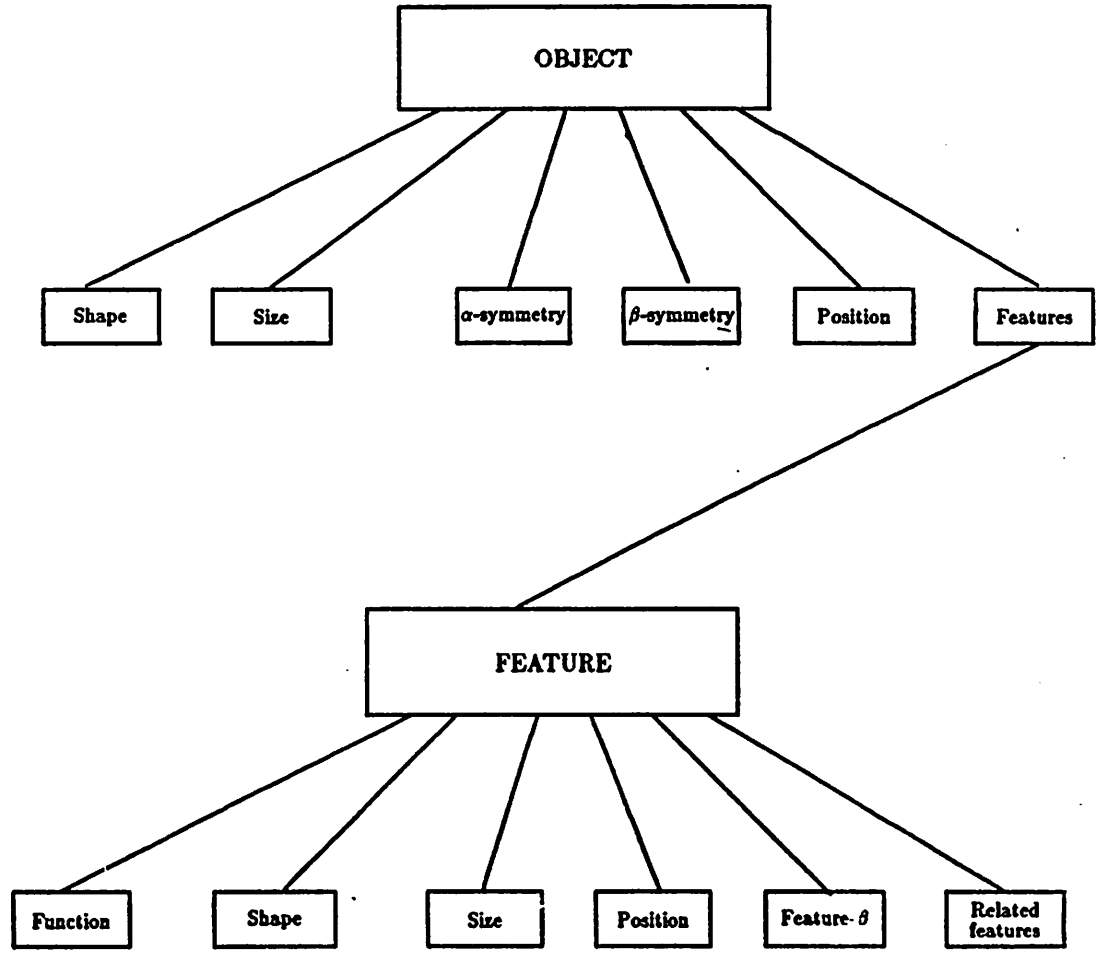


Figure 3: Hierarchical Object Models

Its largest dimension is expressed as the length of the object, the second largest as its width and the smallest as its height.[1]

Characterization of object symmetry is based on the notions of the *principal axis* of an object and rotational symmetry of an object about a specified axis. Boothroyd [6] defines the principal axis of an object to be the axis perpendicular to the cross-section of its degenerated envelope and passing through the centroid of the degenerated envelope. The degenerated envelope of an object is defined as the smallest cylinder, regular prism or rectangular prism encloses the object except for small protrusions.

The rotational symmetry of an object about a specified axis in space represents the smallest angle greater than zero, through which the object can be rotated about the axis and repeat its orientation. The following provides a precise definition of rotational symmetry about a specified axis: Given an object O, and an axis a in 3-space, let SG denote the subset of the interval $(0, 2\pi]$ such that for each x in SG, rotating O about a by x yields identity, that is, leaves the orientation of O unchanged. The rotational symmetry of O about the axis a is given by the infimum of the set SG. When SG is finite, the object is said to have finite symmetry about the axis a, and it is said to have infinite symmetry, otherwise.

Given an object, and a specification of its principal axis, the β-symmetry of the object is defined to be the rotational symmetry of the object about its principal axis. For example, the β-symmetry of a cylinder would be 0, because the principal axis of a cylinder coincides with the axis of the cylinder, and the cylinder has infinite rotational symmetry about its axis. The α-symmetry of an object is defined to be the rotational symmetry of the object about an axis perpendicular to its principal axis. For the cylinder discussed above, the α-symmetry would be π [6]. The object models require the specification of rotational symmetry about two arbitrary axes in the α-symmetry and β-symmetry attributes, encoding explicit specification of the axes and angle of symmetry.

A right-handed coordinate frame is embedded within each object. The origin of the coordinate frame will generally lie at the center of mass of the object with the X, Y,

[1] A different naming convention could be used provided the procedures that operate on these representations are appropriately modified.

and Z-axes of the frame along the length, width and height of the object.[2] Features are specified in terms of the object coordinate frame, circumventing the problem of having to update the position of each feature when the object is moved. Thus, the position attribute of an object is given by a 4x4 homogeneous transformation matrix, specifying the location of the origin and orientation of the axes of the object coordinate frame in terms of the workspace coordinate frame [44].

Features of the object that are identified to be of relevance to assembly processes, such as cylindrical hole, thread, etc., are labeled and associated with the object. Specific information about the features and their interrelationships is represented in subsequent levels of the hierarchy.

4.3 Feature Representation

As indicated in Figure 3, a set of attributes is associated with each feature of an object, analogous to the object-attributes associated with each object. Units of information relating to the features are termed *feature-attributes*. The feature-attributes that are included in the object models are:

- function
- shape
- size
- position
- feature-β
- sub-features
- feature relationships.

The function associated with an assembly feature may take one of two values: *insertor* or *container*, depending on whether the feature has an overall convex shape or a concave shape.

[2]These choices for origin and axes of the object coordinate frame are only a possible set of choices and not necessary ones. The basic scheme works even if a different origin and orientations for axes are chosen.

Table 1: Shapes and Corresponding Size Parameters for Features

Shape	Size Parameters
Cylindrical hole:	internal-diameter of the cylinder depth of the hole
Rectangular hole:	length of the rectangle width of the rectangle depth of the hole
Thread:	whether the thread is internal or external diameter width of the thread pitch of the thread
Cylindrical shaft:	diameter of the cylinder height of the cylinder
Rectangular plane face:	length of the rectangle width of the rectangle
Circular plane face, Semi-circular face:	diameter of the circle
Cylindrical surface, Semi-cylindrical surface:	height of the cylinder diameter of the cylinder
Semi-cylindrical depression:	diameter depth

The model does not restrict the range of objects handled by an assembly planning system it supports, in terms of the shape of objects. However, such an assembly planning system is delimited by the classes of features supported by the model. Although the set of features included in the model are extensible, the current model as implemented in the ASSEMBLE system is only equipped to handle the types of features listed in Table 1, where feature types are classified according to their shape. The dimensional (size) information required about features varies depending on the shape of the feature and is described in Table 1 for each of the feature types.

The position of each feature is described by specifying the *center* of the feature and the direction of its *normal axis* in terms of the object coordinate frame. The definitions of center and normal axis for each feature type listed in Table 1 are provided in Figure 4. The normal axis always passes through the center of the feature and the direction of the normal axis is away from the object.

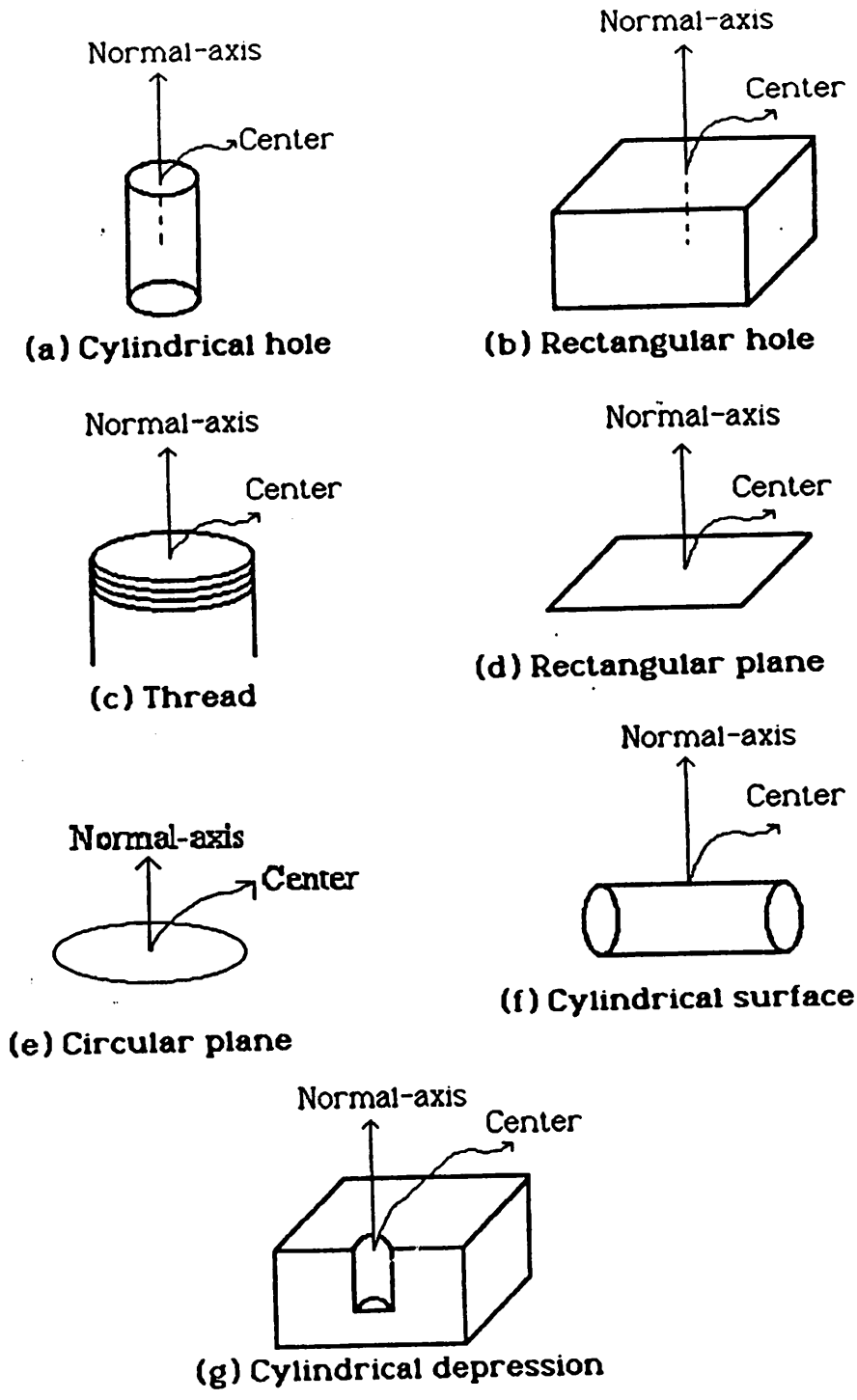


Figure 4: Definition of Center and Normal Axis for Feature Types

The β-symmetry property defined for objects can be applied to features of an object as well. Thus, feature-β is used to denote the rotational symmetry of a feature about the normal-axis of the feature. For example, the feature-β value of a cylindrical hole will be 0°, that of a rectangular hole will in general be 180°(but 90° for a square hole), and that of a hexagonal bolt head will be 60°.

The subfeatures of a feature represent the constituent surfaces of a feature, such as the cylindrical face and the plane bottom face of a cylindrical hole or shaft. The subfeature attributes included in the model are:

- type
- shape
- dimensions
- position.

The class of subfeatures encountered by ASSEMBLE is limited by the types of features handled by the system (listed in Table 1). The type attribute associated with a subfeature takes one of two values: *cylindrical_face* or *plane_face*. The shapes associated with subfeatures are a subset of the classes of shapes associated with features, such as cylindrical surface, rectangular plane face and circular plane face. The dimension and position information are represented and interpreted exactly as the corresponding attributes of features. Subfeatures and their attributes play a key role in the spatial reasoning component of an assembly planning system. The subfeature attributes are derivable from the feature information included in the model. So, in ASSEMBLE, the subfeatures are represented explicitly only when a feature is identified to be a mating feature. The process of generating the subfeatures as well the derivation of their attributes as implemented in ASSEMBLE are described in detail in section 6.6.2.

4.4 Feature Relationships

The information relating to features falls into two categories: geometric and topological. On the one hand, there is shape, size, and position information required to define

the geometry of a feature. On the other hand, there is information pertaining to how the features of a single object or subassembly are positioned with respect to each other. This is expressed as relationships between the features of an object or subassembly. The feature relationships currently defined are:

- *adjacent*,
- *overlapping*,
- *included*,
- *encompassing*,
- *coaxial*.

These relationships are illustrated by using the example (due to Rafael Huber, cf. [61]) shown in Figure 5, which describes the components for assembling a light bulb fixture. Two features of an object are *adjacent* when the surfaces of the object that define the feature are adjacent to each other, i.e., they share a common edge. In object A of the example shown in Figure 5, features A3 (which is a cylindrical hole) and A4 (which is an external thread) are adjacent as they share a cylindrical edge. Two features of an object are *overlapping* when the surfaces defining these features are overlapping. In object D of the example above, features D1 (a cylindrical hole) and D3 (which is an internal thread on the inside surface of the hole) are overlapping features. A feature f_i is *included* in another feature f_j of an object O when the surface defining f_i is completely contained in the surface defining f_j. The *encompassing* relation is the inverse relation of the *included* relation so that whenever f_i is *included* in f_j, f_j is *encompassing* f_i. Two features of an object or subassembly are *coaxial* when their normal axes are collinear. For example, in the example shown in Figure 5, features A3 and A4 of object A are *coaxial*. The relations *adjacent*, *overlapping* and *coaxial* are symmetric relations while *included* and *encompassing* are inverses of each other.

Conceptually, the features of an object may be represented as a graph, where each feature is represented as a node, and a relationship existing between two features is represented by an arc, appropriately labeled, between the corresponding nodes. We will refer to this entity as the *feature graph* of an object. The feature graph of object A of the light bulb fixture assembly of Figure 5 is shown in Figure 6.

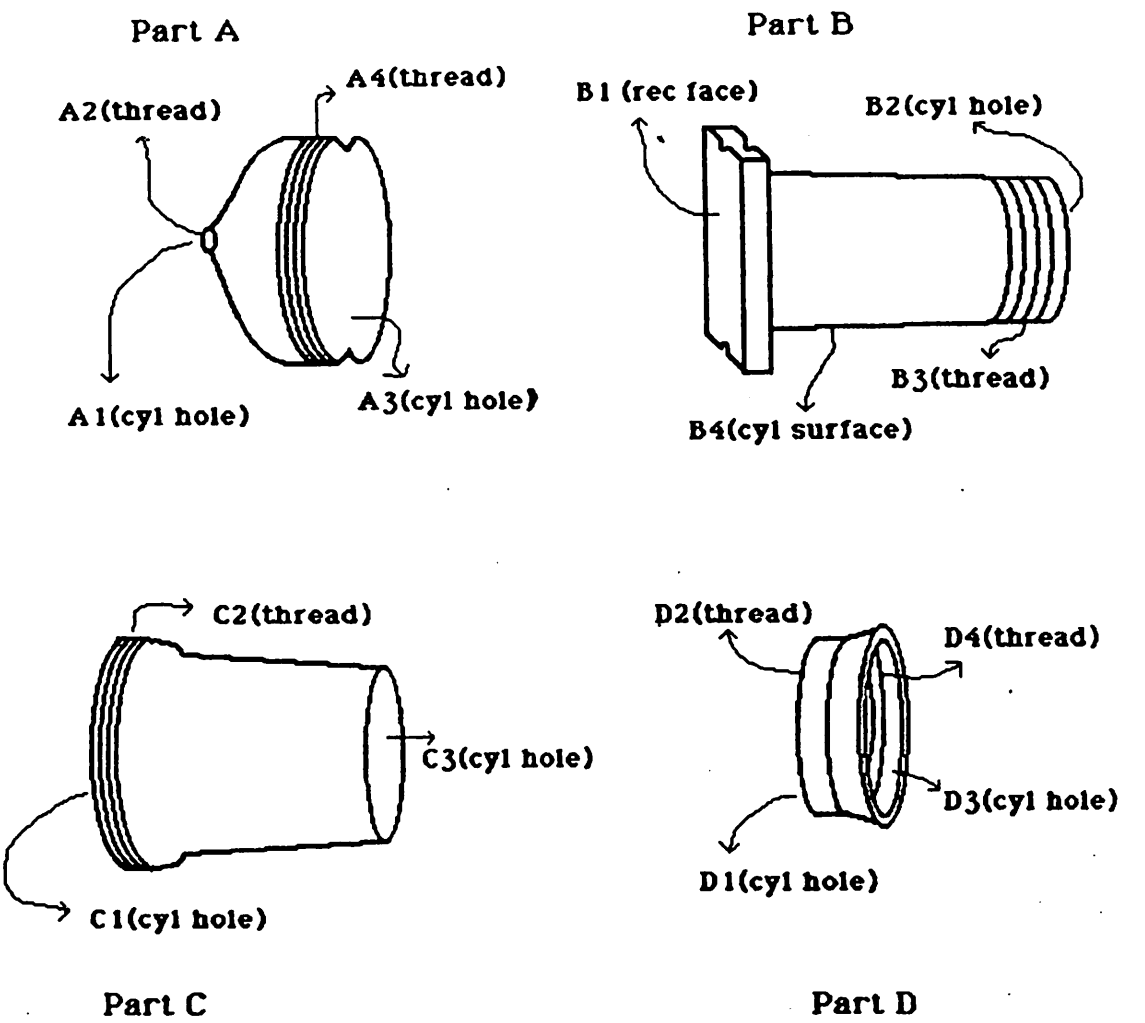


Figure 5: Assembly of a Light Bulb Fixture

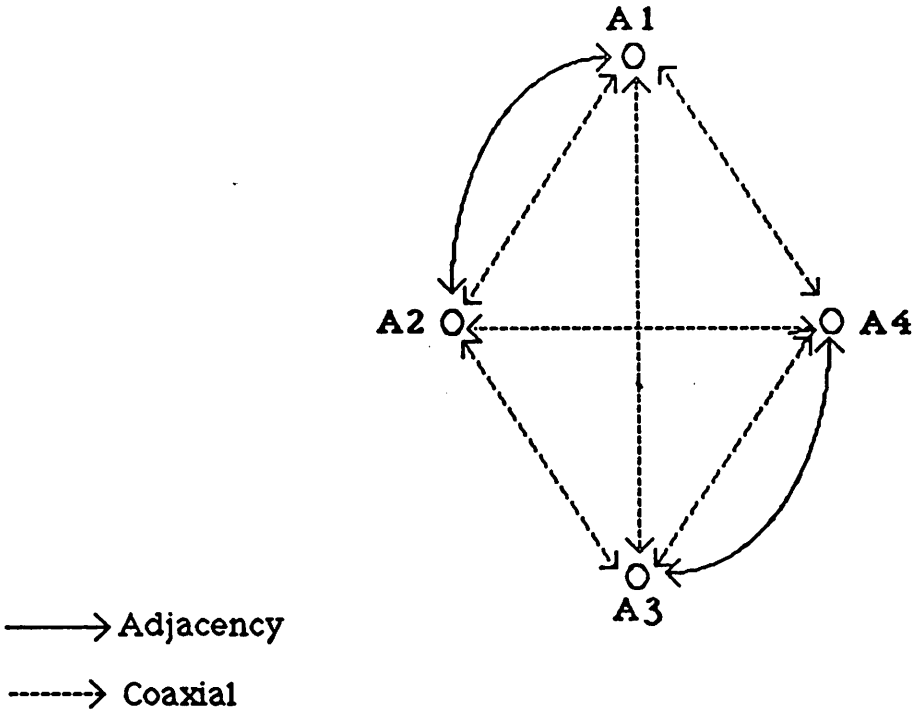


Figure 6: Feature Graph of Object A of Light bulb Fixture Example

While topological information can be deduced from the positional information, it is useful to represent this explicitly, in order to be able to infer the possible effects of operations involving one feature on related features of the same object or subassembly.

Aside from these rather loose specifications of relative positions between features defined by these relationships, strict relative positions between features may be obtained from the position specifications of features with respect to the object coordinate frame.

4.5 Related Work

Typically, an assembly task is described by specifying a homogeneous transformation matrix for each object that comes into contact with another. The RAPT language [45] describes assemblies in terms of set of symbolic spatial relationships between features. For example, insertion of a cylindrical peg into a cylindrical hole is given by a *fits* relationship between the cylindrical surfaces of the peg and the hole and an *against* relationship

between the bottom surfaces of the peg and the hole. There is an inference engine which then interprets these symbolic spatial relations to arrive at a homogeneous transformation matrix describing the position and orientation of a coordinate frame associated with the peg relative to a coordinate frame embedded in the hole. The specification of the assembly in the assembly planning system described in this dissertation is at a higher level. Therefore, specification of an assembly in terms of spatial relationships, as in RAPT, is an intermediate representation that is derived automatically by the assembly planning system.

LAMA [31] is an early attempt at a task planning system that focuses on assembly tasks. It uses an object description that is based on Constructive Solid Geometry (CSG). Thus, objects are described as unions of primitive objects, such as cuboids and cylinders as solids and holes. LAMA uses a hierarchical representation of objects by adopting the notion of subparts of objects. For example, a piston is described as being composed of a piston-body, piston-ridge and pin-hole where the piston-body is further decomposed into a piston-cylinder and piston-cavity.

With each subpart is associated its size, type and relative position. All subparts are approximated as rectangular or octagonal right prisms. The subparts concept of LAMA is quite akin to the notion of features adopted in this work except that LAMA's decomposition into subparts decomposes the volume of the object while the features that are a part of the object models described in this chapter decompose the surface of the object. LAMA's representation does not include the notion of relationships defined between features of the same object. However, the work described in [31] starts with a detailed assembly plan where information on the goal position and orientation of one object relative to another is specified. Thus, the object representations are used primarily for problems such as grasp point selection and collision detection.

AUTOPASS [28,63] is another attempt at a high-level programming system for computer controlled mechanical assembly. The system includes a model of the assembly environment called the *world model*. The world model is represented as a graph structure in which each node represents a part, sub-part or assembly. Specific semantics are associated with each type of node in the graph. A *part* is a three-dimensional object whose shape is fixed but whose position and orientation may be changed dynamically.

A *sub-part* is a volumetric entity out of which parts are composed. An *assembly* (or *aggregate*) is composed of parts or assemblies that bear a fixed functional relationship to each other; however, the internal geometry of the assembly may change. The arcs of the graph represent one of four kinds of relationships that exist between the nodes: *part-of, attachment, constraint,* and *assembly-component.* AUTOPASS proposes a representation for objects that encompasses most of the object attributes included in the object models described in this chapter. It also provides a uniform treatment of objects, sub-parts (features) and assemblies. The relationships provided in AUTOPASS are of a very different class from those described in this work. While the object models described here define relationships on features, describing their topological relations and symmetry properties, AUTOPASS defines relationships primarily between object and assemblies to characterize kinematic and stability relationships between objects. In ASSEMBLE, such kinematic constraints are inferred from knowledge about the operations involved in the assembly, and issues of stability are not addressed.

Lee and Gossard [26] describe an assembly using the symbolic spatial relations of RAPT, while using boundary representation for describing the shape of objects. A notion of assembly features is used and spatial relationships are defined in terms of these features. Like RAPT, the work confines itself to arriving at the relative positions and orientations of objects, although a numeric solution to the equations obtained through the symbolic spatial relations is sought as opposed to symbolic solutions as in RAPT.

4.6 Summary

Common to all of the representation discussed above and the one detailed in this chapter is the decomposition of objects into some component features. The representation described here adopts a surface decomposition approach similar to RAPT and Lee and Gossard's work while LAMA and AUTOPASS use a volumetric decomposition. What distinguishes this work from others is the problem for which the representation is being used, namely, that the specification of the assembly is at a higher level than those used in other systems and some problem solving is required to generate reasonable hypotheses about the relative positions of objects implied by the assembly. The prob-

lem solving ability hinges upon the information the system has about the objects and assembly operations that make up an assembly. This work attempts to determine the key characteristics of an object that are relevant for the problem of planning an assembly from such high-level specifications. The experimental assembly tasks processed by ASSEMBLE are discussed in Chapter 7. They serve to test the efficacy of the object models described here.

The accessibility analysis component of ASSEMBLE reasons about the accessibility of features based on prior operations. The feature relationships represented in the *feature graph* and the interpretations of assembly operations are the essential ingredients to perform this reasoning. However, the treatment of geometric reasoning in the current version of ASSEMBLE is not sufficiently general to enable the development of models for the subassembly that are at the same level of detail as the models of the objects themselves.

The representation of objects in the proposed model requires each object in the workspace to be described in detail individually. However, assembly situations often involve a number of identical objects or objects that belong in the same class (the controller assembly described in Section 7.5 illustrates this observation). For example, bolts of different sizes still have the same set of features and topological relations between features although the dimension of these features may differ. Thus, object models need to be extended to support the notion of classes of objects extending the hierarchy one level upwards. Physical objects present in the workspace can then be viewed as specific instances of object classes.

Planning an assembly task requires reasoning about object interference in addition to spatial relations among objects. However, the information present in the object models cannot adequately support such reasoning. In order to include reasoning about object interference, the object models need to be augmented to reflect volumetric characteristics of objects as well. Thus, what has been described here is but a component of object information that is required for the purposes of reasoning about assembly.

CHAPTER 5

PROBLEM DECOMPOSITION

In this work, the assembly planning problem is viewed as comprising the determination of the positions and orientations of component objects relative to each other and the determination of the robot motion primitives required to achieve these relative positions. This chapter details the problem decomposition and reasoning processes for solving the assembly planning problem given the object models detailed in Chapter 4 corresponding to the objects involved in the assembly.

The first part of the problem is to derive the transformation matrices that describe the relative locations of objects in the final assembly configuration. Recall that the input provided to the assembly planning system is a sequence of instructions that specify the assembly operations and objects involved for each of the operations. From the specification of the operations and the object information provided by the object models, many of the missing details need to be deduced in order to express the assembly operations in terms of relative locations. The system achieves this by refining the problem statement successively, with each refinement providing more detail than the previous level. The levels of refinement correspond to the levels in the object model hierarchy. The abstraction hierarchy is shown in Figure 7.

Thus, the description of the problem at the object level is first refined to a feature level description. This refinement process is achieved through the use of geometric constraints associated with the operations, symmetry properties of objects, assembly heuristics and propagation of accessibility constraints. Feature level descriptions are refined into a set of spatial relationships between geometric entities associated with the features. These spatial relationships are derived by combining information regarding the shapes of the features with the essential kinematic relations suggested by the assembly operation. The

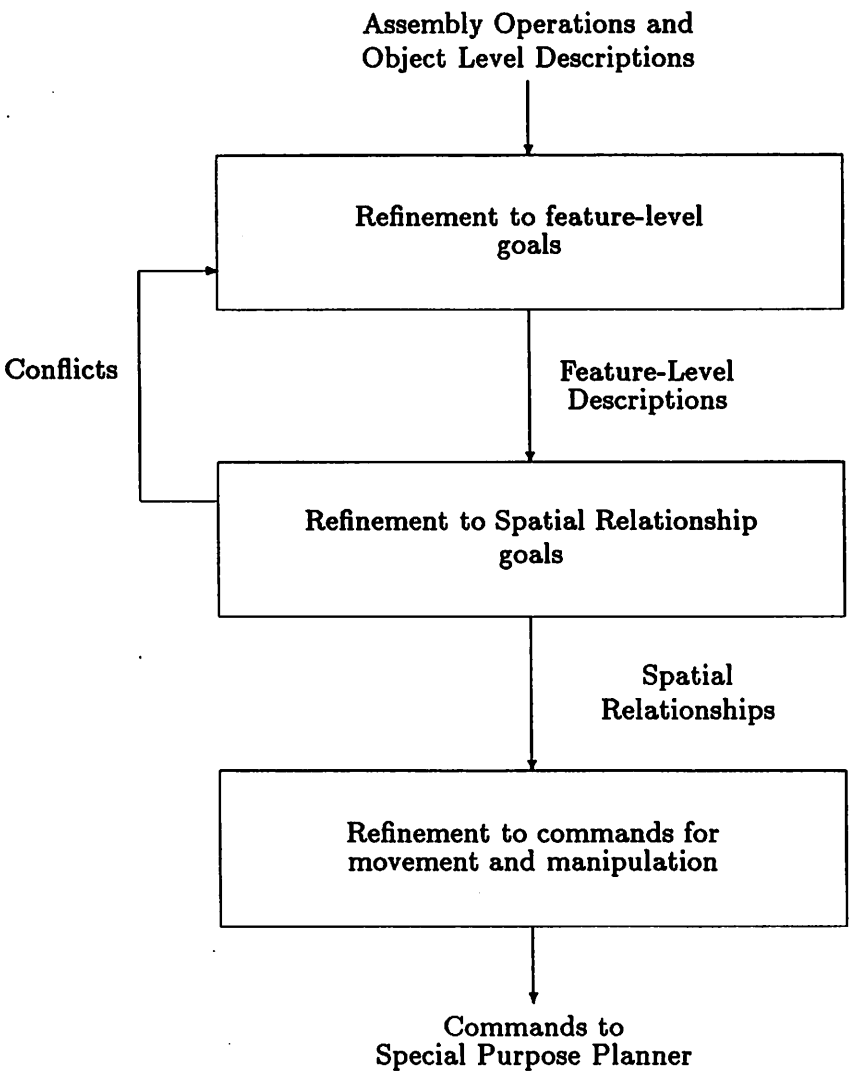


Figure 7: Levels of Problem Refinement

intermediate representation of the problem as a set of spatial relationship goals allows the assembly planner to employ a spatial reasoning system. This enables the assembly planner to identify whether the choices made in the course of problem refinement thus far are viable and provides a way to combine the spatial relationships to obtain tighter relationships. The spatial relationships are then refined into the necessary grasp, free-motion and fine-motion commands.

5.1 Transformation to Feature-Level Descriptions

The primary task of the assembly planner is the determination of the mating features for each of the operations specified in the input. Associating a finite set of features relevant for assembly with each object, as in the object model, discretizes the problem. The set of possibilities an assembly planning system needs to consider for each assembly instruction is the cartesian product of the set of features associated with each of the objects specified in the assembly instruction. The DOUBLE-INSERT operation generates a larger candidate set, the cartesian product of the cartesian product set with itself, as two pairs of mating features are required. This set of possibilities is reducible by consideration of several sources of constraints specific to the domain of assembly. These are discussed below.

Each assembly operation brings together a pair of mating features, or two pairs of mating features as in the case of the DOUBLE-INSERT operation. Each feature of the pair has a specific role in the accomplishment of the operation. For example, an INSERT operation mates a convex feature with a concave feature; a THREAD operation mates an externally threaded feature with one that is threaded internally; a PLACE operation brings together two convex features; and a DOUBLE-INSERT operation mates a pair of convex features in one object with a pair of concave features in the other object. This information is explicitly represented in the *function* attribute of features and is also easily derived from the shape information about features.

Besides the requirement that the *function* attributes of a pair of mating features be complementary for a specified assembly operation, the operation also provides some

constraints on the dimensions of the features. For an INSERT operation, the concave feature should volumetrically enclose the corresponding convex feature. In the case of a THREAD operation, the threaded surfaces should have nearly equal diameters and the width and pitch of the threads should match. For a DOUBLE-INSERT operation, features of each pair have to satisfy the dimensional constraints as in the INSERT operation with additional constraints on the relative distance between the two features on each object as well as the relative orientation of their normal axes.

All of these constraints, arising from the specification of the assembly operation, are verifiable given the information contained in the object models. They serve to eliminate pairs of mating features which are inappropriate for the assembly operation specified. The verification of the volumetric enclosure requirement can be made more precise by using a solid modelling system. As noted in Chapter 4, the emphasis in the development of object models has been on the surface characteristics of objects and features.

Another source of information that an assembly planning system can utilize to reduce the candidate set of mating features is the symmetry of objects. When an object has finite symmetry, features that may be labeled distinctly and occupy distinct locations in the object may be indistinguishable for the purposes of assembly. The α-symmetry and β-symmetry attributes associated with objects contain information about the rotational symmetry of objects about specified axes. For example, a cylindrical peg has a rotational symmetry of π about any axis perpendicular to the axis of the cylinder, which is the α-symmetry of the cylinder; either end of the peg can be used for insertion. Based on the α-symmetry (β-symmetry) of an object, the relation, *α-equivalence* (*β-equivalence*), on the features of an object is defined as follows: Let f_i and f_j denote two features of an object O, whose positions with respect to the world coordinate frame are given by L_i and L_j, respectively. If the object is placed such that the object coordinate frame coincides with the world coordinate frame, then L_i and L_j will precisely be the position specification of features f_i and f_j with respect to the object coordinate frame. Let L_i' and L_j' denote the resulting positions of features f_i and f_j with respect to the world coordinate frame after rotation of the object O about the specified axis of symmetry by an angle θ $(0 < \theta \leq \pi)$, where θ is a multiple of the angle representing α-symmetry (β-symmetry). The features f_i and f_j of O are said to be *α-equivalent* (*β-equivalent*) if

$L_i' = L_j$ or $L_j' = L_i$ for some θ. An ability to deduce the symmetry relationship existing between two features of an object which are both being considered as possible mating features, can enable an assembly planning system to choose one of them, arbitrarily, and eliminate pairs of mating features that involve the other.

When several possible pairs of mating features exist, heuristics that relate specifically to assembly operations can be used to evaluate if one option is more plausible than another. As different heuristics may favor different options, the assembly planning system requires a system of ranking the options based on the heuristics. The specific heuristics that are part of ASSEMBLE and the mechanism used in the system for ranking the alternatives are described in Chapter 6.

Finally, the order in which the assembly operations are to be performed, which is specified as input to the system, allows for reasoning about the accessibility of features of an object, when the object has already been used in a prior operation. The feature relationships encoded in the object model facilitate reasoning about accessibility. The basic premise is that if two features are unrelated (that is, they are not adjacent, overlapping, coaxial, and do not have an included-encompassing relation between them) use of one feature for an earlier operation will not affect the use of the other for a subsequent relation. When such a relationship exists, however, the symbolic descriptions of relationships are inadequate to conclusively determine, in all cases, whether or not there will be interference. This limitation generates the need for other systems of verification to ensure the viability of a proposed set of mating features.

5.2 Feature-Level Descriptions to Spatial Relationships

The process of assembly deals with bringing surfaces of objects in contact with each other. Determining the mating features of objects identifies which surfaces are to come into contact. The reasoning processes suggested at the feature level, with the exception of the accessibility analysis process, all operate locally on an assembly operation. The mating features suggested by these processes can be spatially inconsistent, globally.

Another essential characteristic of an assembly operation is that it constrains certain

motion degrees of freedom of one object relative to another. The precise nature of these constraints is a function of the assembly operation as well as the geometry of the features involved. For example, placing one object on another constrains translational motion along one axis and rotational motion about the other two. For the purposes of this discussion we will assume that the constrained degrees of freedom are translation along the Z-axis and rotation about the X- and Y-axes. Insertion of a cylindrical peg into a hole implies that the bottom surfaces of the peg and hole will be against each other (which results in the same constraint as above) with the additional requirement that the axes of the peg and hole are aligned. Keeping the same frame of reference as described above, this additional requirement constrains translation along, and rotation about, the X- and Y-axes. Combining the two sources of constraints results in one degree of freedom, namely, rotation about the Z-axis. The geometry of the features also plays a role which is illustrated by the case where a square peg is inserted into a square hole; five planar contacts are generated resulting in all six degrees of freedom being constrained.

The use of such constraints to describe assemblies is not novel. The RAPT system, which will be described in more detail in the next section, describes assemblies by symbolic representations of such constraints. Morris and Haynes[40] propose such an approach for describing assemblies. This work adopts the same idea and deduces symbolic spatial relationship constraints based on the specified assembly operations. In the remainder of this section, an analysis of the assembly operations included in this work, namely, INSERT, THREAD, DOUBLE-INSERT and PLACE, are presented and the spatial relationships implied by these operations are derived.[1] Based on this analysis, a working set of spatial relationship predicates are compiled which provide the vocabulary for the intermediate representation of assembly tasks in terms of spatial relationships between features of objects.

---

[1]The DOUBLE-INSERT operation does not require a separate analysis. The two pairs of mating features generate two sets of constraints as if an INSERT operation is being performed with each of those pairs of features.

5.2.1 Analysis of Assembly Operations

The following notation is used to denote the position attributes of features:

- $center_j^i$ is a 3-tuple specifying the center of $feature_j$ with respect to the coordinate frame in $object_i$.
- $(x_j, y_j, z_j)^i$ or equivalently $(x_j\mathbf{i} + y_j\mathbf{j} + z_j\mathbf{k})^i$ specifies the unit vector along the direction of the normal-axis of $feature_j$ with respect to the coordinate frame in $object_i$ where $\mathbf{i},\mathbf{j},\mathbf{k}$ are unit vectors along the x, y and z axes of the coordinate frame, respectively.

The derivation of spatial relationships implied by the assembly operations, presented below, relies on certain assumptions about the geometry of the mating features.

INSERT:

Consider an INSERT operation such as

Insert $feature_k$ of $object_i$ into $feature_l$ of $object_j$

An INSERT operation may correspond to a *close-fit* where the dimensions of $feature_k$ and $feature_l$ are nearly equal, or it may correspond to a *loose-fit*. In the case of a *close-fit*

- the axes of $feature_k$ and $feature_l$ will be opposed; i.e. their normal axes will be collinear and opposed in direction.
- the bottom surface of the feature being inserted, $feature_k$, will be against the bottom surface of the container, $feature_l$.
- If the insertion and container features are cylindrical, then their cylindrical surfaces will be in contact. If they are rectangular then the surfaces on the sides will be in contact.
- The center of the insertor feature, $center_k^i$ will be at a distance equal to the depth of the hole ($feature_l$) away from the center of $feature_l$, $center_l^j$, along the direction $(x_l, y_l, z_l)^j$.

Successful completion of the *close-fit* operation described above for the case of a cylindrical shaft ($feature_k$) in a cylindrical hole ($feature_l$) can then be expressed as the satisfaction of the following spatial relationship goals:

1. opposed$((x_k, y_k, z_k)^i, (x_l, y_l, z_l)^j)$
2. against($feature_k.bottom$,$feature_l.bottom$)
3. fits($feature_k.curved\text{-}face$,$feature_l.curved\text{-}face$)
4. $center_k^i = center_l^j - depth_l * (x_l, y_l, z_l)^j$.

In the case of an INSERT operation corresponding to a *loose-fit*, three constraints may be derived. Constraint (3) listed above will not apply. Also, constraint (1) is modified to express a weaker constraint, namely that the normal axes of $feature_k$ and $feature_l$ should be parallel and opposed in direction. (They need not be collinear.) Thus, the spatial relationship goals for a *loose-fit* insert operation will be as follows:

1. xparallel$((x_k, y_k, z_k)^i, (x_l, y_l, z_l)^j)$
2. against($feature_k.bottom$,$feature_l.bottom$)
3. $center_k^i = center_l^j - depth_l * (x_l, y_l, z_l)^j + \delta$ where δ represents an arbitrary translation in the plane perpendicular to $(x_l, y_l, z_l)^j$.

THREAD:

Consider a THREAD operation such as

Thread $feature_k$ of $object_i$ into $feature_l$ of $object_j$

where $feature_k$ is threaded externally and $feature_l$ is threaded internally. Successful completion of the operation is expressed as the satisfaction of the following three spatial relationship predicates:

1. opposed$((x_k, y_k, z_k)^i, (x_l, y_l, z_l)^j)$
2. fits($feature_k.curved\text{-}face$,$feature_l.curved\text{-}face$) fits
3. $center_k^i = center_l^j - (min(width_k, width_l) * (x_l, y_l, z_l)^j)$.

PLACE:

Consider a PLACE operation such as

Table 2: Proposed Spatial Relationship Predicates

Spatial Relations
opposed(axis1, axis2)
aligned(axis1, axis2)
parallel(axis1, axis2)
xparallel(axis1,axis2)
fits(feature1,feature2)
against(feature1, feature2)
coplanar(feature1, feature2)

Place $feature_k$ of $object_i$ on $feature_l$ of $object_j$

The spatial relationships that will be in effect by accomplishing this operation will be:

1. xparallel($(x_k, y_k, z_k)^i$, $(x_l, y_l, z_l)^j$)
2. against($feature_k$, $feature_l$)
3. $center_k^i = center_l^j + \delta$ where δ represents an arbitrary translation in the plane perpendicular to $(x_l, y_l, z_l)^j$, that is, the plane representing $feature_l$.

There are similarities between the spatial relationships implied by the *close-fit* INSERT operations and the THREAD operations as well as between the spatial relationships derived from the *loose-fit* INSERT operations and PLACE operations.

5.2.2 Intermediate Language of Spatial Relationships

Based on the analysis of operations discussed above, a set of spatial relationship predicates are proposed to provide a sufficiently rich vocabulary using which relationships between features of different objects can be specified. The predicates are described below and are summarized in Table 2:

opposed*(axis1, axis2)*: this predicate is true when *axis1* and *axis2* are collinear and opposed in direction. It is implicit that *axis1* and *axis2* are specified with respect to different coordinate frames.

aligned*(axis1, axis2)*: this predicate has an interpretation identical to the **opposed** predicate described above except that it is true only when *axis1* and *axis2* have the same direction.

parallel*(axis1, axis2)*: this predicate is true when *axis1* and *axis2* are parallel and have the same direction.

xparallel*(axis1, axis2)*: this is similar to the **parallel** predicate above except that it is true only when *axis1* and *axis2* have opposite directions.

fits*(feature1, feature2)*: this predicate applies to features which are cylindrical surfaces. The predicate evaluates to true when axes of the cylinders are collinear and when the cylindrical surfaces are in contact.

against*(feature1, feature2)*: this predicate applies when both features are plane surfaces or when one is a cylindrical surface and the other a plane surface. The predicate evaluates to true when *feature1* and *feature2* are in contact with their normal axes opposed in direction.

coplanar*(feature1, feature2)*: this predicate applies to features that are plane surfaces. The predicate evaluates to true when the extensions of *feature1* to their corresponding infinite surfaces coincide with their normals pointing in the same direction.

Describing the assembly operations in terms of spatial relationships provides a significant source of constraints. The same objects are involved in several assembly operations. Combining the spatial relationship constraints arising from the specified sequence of operations will allow an assembly planning system to determine whether the choices made at the feature level are feasible. Secondly, combining the spatial relations allows for determining tighter relationships between objects. Typically, assembly tasks give rise to closed kinematic chains so that combining these spatial relationship constraints is sufficient to determine the pose of objects relative to each other in the final assembly configuration. The problem of classifying and combining spatial relationships was addressed by the RAPT system, which is described in the following section. The ASSEMBLE system interfaces with RAPT to avail itself of this analysis capability.

5.3 The Spatial Reasoning Component

The RAPT system comprises a language for describing assemblies in terms of symbolic spatial relationships between features of objects and an associated inference system that

works with these symbolic spatial relationships. The features RAPT deals with are at the level of subfeatures defined in the object models. In the rest of this section, any reference to the term *feature* will refer to a RAPT feature as opposed to the assembly features defined in this work. The language and the inference machinery are broader in scope than is required for the assembly planning problem addressed in this dissertation. The RAPT language allows assembly operations to be specified partly in terms of spatial relationships to be established between features of objects and partly in terms of how one object is to move with respect to some feature of another. Thus, the system has the capacity to represent an assembly task as a sequence of intermediate states, where each state gets described in terms of symbolic spatial relationships. In this work, the analysis is confined to the final assembly configuration.

As in the ASSEMBLE system described in Chapter 6, RAPT defines a coordinate frame in each object and describes the position of the object as a 4x4 homogeneous transformation matrix. A coordinate frame is associated with each feature with the convention that the X-axis of the frame is normal to the surface, pointing away from the object (see Figure 20). These coordinate frames associated with features are defined with respect to the coordinate frame embedded in the object.

The main input relationships RAPT deals with are *fits* and *against*. A *fits* relationship is used to describe two cylindrical surfaces in contact such as a shaft in a hole. An *against* relationship is used to describe contact between two planar surfaces, a planar surface and a cylindrical surface, a planar surface and a spherical surface, or two spherical surfaces. RAPT treats surfaces to be of infinite extent. Therefore, saying that a shaft *fits* a hole implies that the infinite cylindrical surfaces obtained by extending the shaft and the hole about their respective axes will be in contact, provided they have the same radii.

Internally, the system represents 12 distinct relationships between features of objects. These are *fits, agpp, agpc, agcp, agps, agsp, agss, lin, rot, parax, fix*, and *linlin*. The relations *agpp* through *agss* in this list stand for against relationships where *p* stands for a planar feature, *c* stands for a cylindrical feature, and *s* stands for a spherical feature. The relations *lin, rot, parax, fix*, and *linlin* are relationships derived by the system. *Lin* represents a single translational degree of freedom, *rot* represents a single rotational degree of freedom, *parax* represents translational degrees of freedom along all three axes

and a rotational degree of freedom about one of the axes. The *linlin* relation represents two translational degrees of freedom. The *fix* relation represents the fact that one object is completely fixed with respect to the other, that is, there are no relative degrees of freedom.

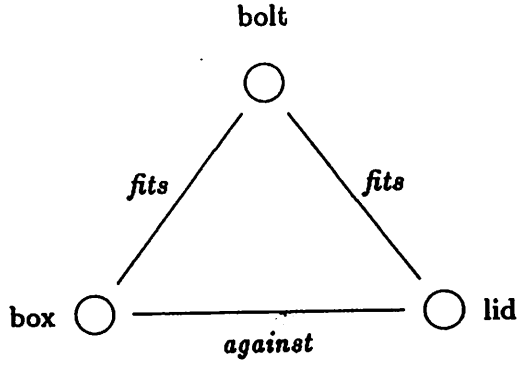
Two inferencing systems were developed for RAPT. The former one, documented in [46], converted the specified spatial relationships between features of objects into kinematic equations and solved them algebraically. It was observed that this system spent a great deal of time performing algebraic computations while the relationships to be combined fell into a relatively small number of stereotypes. A subsequent implementation, described in [47], was developed. The former system was more general, while the latter was found to be more efficient. It is the latter system that is used by ASSEMBLE.

The latter version of RAPT adopts a graph reduction mechanism for the inferencing. The objects in an assembly form the nodes of the graph, and the specified spatial relationships between features of these objects form the set of labeled arcs.

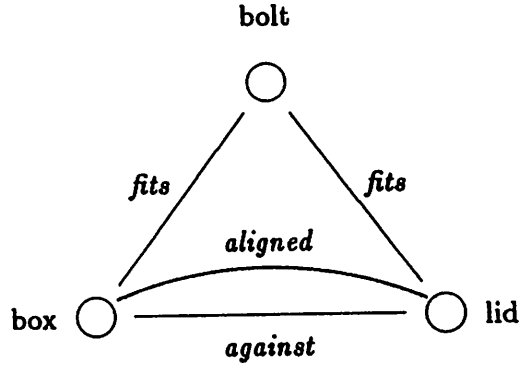
The system works by first attempting to detect cycles of length 2 in this graph that can be replaced with a single arc. An example is the peg in hole scenario which is a combination of *fits* and *against* which can be replaced by a *rot*. Typically, a few different possible spatial relationships may result from combining two existing relationships. Which one of these possibilities applies in a specific situation depends on the relative orientations of the features appearing in the spatial relationships being combined.

Secondly, the system attempts to introduce additional arcs to cycles of length 3, where there is potential for deriving a tighter spatial relationship by reducing the resulting cycle of length 2. An example would be a box, lid and bolt scenario where the lid is placed on the box and the bolt inserted to fasten the box and lid. In this situation, the box and lid are in an *against* relationship, and the bolt is in *fits* relationship with both the lid and the box. The two *fits* relationships can be used to infer an *aligned* relationship between the box and the lid implying that the holes are to be aligned (see Figure 8). The *aligned* and *against* relationships can be combined to produce a *rot* relationship between the box and the lid.

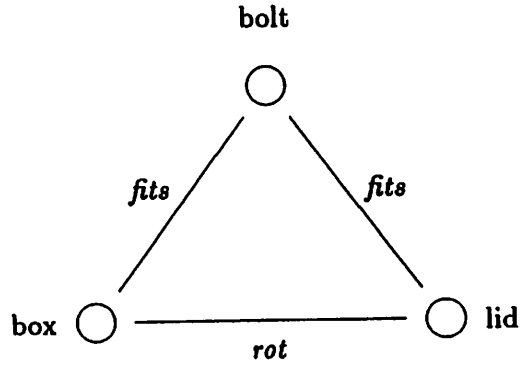
When a *fix* relation is obtained between two objects, the two nodes representing



(a) Initial graph for the box, lid and bolt assembly



(b) Adding an arc to the cycle of length 3



(c) Reducing resulting cycle of length 2

Figure 8: Inferencing using Graph Reduction in RAPT

these objects are merged so that the system treats the two objects as a single object in subsequent reasoning.

RAPT performs its graph reduction by table look up. The system uses two tables. One of these encodes commonly occurring pairs of relationships between features of two objects which can be replaced by a single relationship between possibly new features of these objects. This table is used for reducing cycles of length 2. A second table encodes commonly occurring situations where a relationship between features of objects, say A and B, and a relationship between features of objects B and C, can be used to infer an additional relationship between some features of A and C. This table is used for introducing additional arcs to cycles of length 3.

The experiments detailed in Chapter 7 illustrate the reasoning capabilities and serve to demonstrate the use of RAPT within the ASSEMBLE system.

5.4 Transformation of Spatial Relationship Goals into Commands for Special Purpose Planners

In Chapter 3, the assembly planner is viewed as a component which interfaces with a grasp planner, path planner and fine-motion planner. This section describes the process by which the parameters that are a part of the commands sent to these special purpose planners can be derived.

While the description of the assembly operations in terms of spatial relationships provides a source of constraints useful for verifying the choice of mating features, the information required by the grasp planner as well as the fine-motion planner can be generated from the feature-level descriptions of the assembly operations. Both of these planners require a specification of the normal axis of the mating feature and the operation. The fine-motion planner also requires specification of the extent of motion, which is derived from the information in the object models.

The commands to the path planner, however, are generated from the description of the operations in terms of spatial relationships. Note that the spatial relationship con-

straints resulting from each operation, as detailed in Section 5.2.1, specify an alignment relationship and a contact relationship. The commands to the path planner should be framed such that the alignment is accomplished and contact is subsequently achieved by fine-motion.

5.5 Summary

This chapter has provided the motivations underlying the decomposition of the assembly planning problem into the problem of feature identification and the specification of the problem as a set of spatial relationships. The conceptual basis behind the various reasoning mechanisms used to refine the problem statement to these levels are outlined. All but the final component of the refinement hierarchy, namely the transformations that produce the commands to the special purpose planners, have been implemented in ASSEMBLE. Chapter 6 provides the algorithmic details of the implemented system, while Chapter 7 details the behavior of the system on some experimental assembly tasks.

As noted earlier in this chapter, both the analysis of the reasoning mechanisms in this section as well the implemetations and experiments show that an ability to represent and reason about the volumetric characteristics of objects can significantly extend the capabilities of the system. It will also serve to strengthen some of the existing components, such as the accessibility analysis phase.

CHAPTER 6

THE ASSEMBLY PLANNER

The previous two chapters discussed the hierarchical object models developed and the corresponding hierarchical problem refinement approach devised for the assembly planning problem. This chapter details the computational realization of the representation described in Chapter 4 and the reasoning process described in Chapter 5 through the development of the program ASSEMBLE. ASSEMBLE has been implemented in Common Lisp. Section 6.1 provides detailed specifications of the input to the system. The intended output of ASSEMBLE is at the level described in Section 3.3.1. The following sections provide an overview of the ASSEMBLE system and a description of how the major subsystems of ASSEMBLE work together, its delimitations, the primary data structures used, and the structure and algorithmic details of its individual subsystems.

6.1 Input to the ASSEMBLE System

The input to ASSEMBLE comprises: (a) a set of high-level assembly instructions along with a set of precedence relationships that are known to hold among these instructions; and (b) for each object in the assembly, information required for building the abstract object models described in Chapter 4.

Each of the high-level assembly instructions is of the form

operation *object1* .. *object2*

specifying the assembly operation to be carried out and the objects involved in the operation. Precedence relationships are specified as a set of pairs where each element of a pair is a label associated with one of the assembly instructions. If (a, b) is one of the specified relationships, it is interpreted as "the assembly operation represented by a should precede

(be executed prior to) the operation represented by b." For the purposes of illustration, the input instructions corresponding to the light bulb fixture assembly shown in Figure 5 are presented below:

Assembly Instructions:

1. Insert B into A
2. Thread C with A
3. Thread D with B

Precedence Relations: (1,2) (2,3).

6.1.1 Object Information

The object models are represented as structures in ASSEMBLE using the *defstruct* feature of Common Lisp. Structure templates of objects have slots corresponding to each of the object attributes — shape, length, width, height, volume, surface area, α-symmetry, β-symmetry, position —, and a list of features. The shape attribute is represented symbolically. The α-symmetry and β-symmetry attributes are represented as one-dimensional arrays of length 4, whose first three components specify the direction of the axis of symmetry while the last component defines the angle of symmetry. The position attribute is represented as a 4×4 array where columns 1 through 3 represent the directions of the X, Y, and Z-axes of the object coordinate frame and column 4 represents the position of its origin. Attributes length, width, height, volume, and surface area are represented as numbers. Object A of the light bulb fixture assembly experiment is shown in Figure 9 along with its reference coordinate frame. Figure 10 shows the top-level representation for object A.

Feature templates have slots corresponding to each of the feature attributes — function, shape, dimensions, position, feature-β, and list of subfeatures — and each of the types of relationship defined between features of the same object listed in Section 4.4. The function and shape attributes are represented symbolically. The dimensions attribute is represented as a list of numbers whose interpretation is based on the value of the shape attribute. The position attribute is represented as a 2×4 array where the first row specifies the position of the center of the feature and the second row specifies the direction of the normal axis of the feature. The feature-β attribute is specified as a number denoting

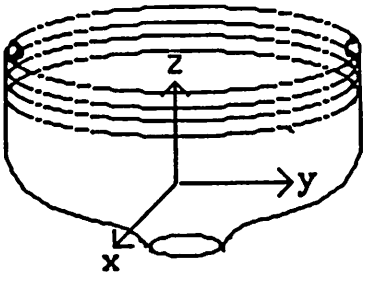


Figure 9: Object A

```
#<object 3436061> is a structure of type object
   NAME:                          part_a
   SHAPE:                         cylinder
   LENGTH:                        3.6
   WIDTH:                         3.6
   HEIGHT:                        2.4
   ALPHA:                         #(0 1 0 6.283185307179587d0)
   BETA:                          #(0 0 1 3.1415926535897936d0)
   POSITION:                      #2A((1 0 0 0) (0 1 0 0)
                                      (0 0 1 0) (0 0 0 1))
   FEATURES:                      (a1 a2 a3 a4)
```

Figure 10: Object Attributes

```
#<feature 3436417> is a structure of type feature
   FEATURE-NAME:                a3
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               cylindrical-hole
   FEATURE-DIMENSIONS:          (3.6 2.0)
   FEATURE-POSITION:            #2A((0 0 1.2 1.0) (0 0 1 0))
   FEATURE-BETA:                3.141592653897936d0
   SUB-FEATURES:                nil
   ADJACENT-FEATURES:           (a4)
   INCLUSIVE-FEATURES:          nil
   ENCOMPASSING-FEATURE:        nil
   OVERLAPPING-FEATURES:        nil
   COAXIAL-FEATURES:            (a4)
```

Figure 11: Feature Attributes

the angle of symmetry.[1] The subfeature information is not required to be specified. Instead, it is derived from the feature information when a feature is hypothesized to be a mating feature. Section 6.6.2 provides a description of the representation of subfeatures and examples are provided in Figure 19. The feature relationships are represented by specifying for each feature f, and each type of relationship r, the list of features g such that f is in relationship r with g. Figure 11 displays the structure representation corresponding to feature A3, the larger cylindrical hole, of object A of the light bulb fixture assembly shown in Figure 5.

6.2 Overview of the ASSEMBLE System

The ASSEMBLE system, when given a task specification and object information as detailed in the previous section, is designed to proceed in a series of steps to generate the desired output commands. The components and the order of their invocation is shown in Figure 12.

As a first step, ASSEMBLE sets out to generate hypotheses about what the mat-

[1]The axis of symmetry is implicitly defined as the normal axis of the feature.

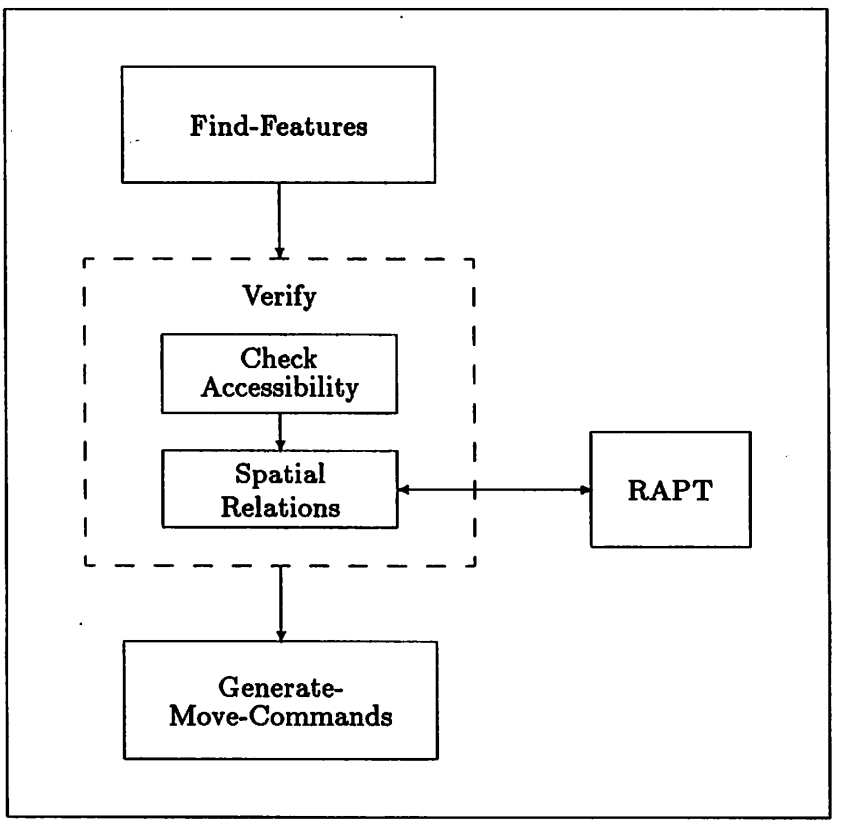


Figure 12: Structure of the ASSEMBLE System

ing features might be, corresponding to each of the assembly operations. It employs the FIND-FEATURES subsystem to generate such hypotheses. The FIND-FEATURES subsystem works by first determining the feasible mating features corresponding to each operation by applying geometric constraints imposed by the operations. Secondly, it identifies equivalent pairs of mating features by reasoning about the symmetry of objects, and reduces any redundancy that may exist in the hypothesis set resulting from object symmetry. Finally, it ranks the possible mating features corresponding to each operation by application of heuristic information. Thus, the result of the FIND-FEATURES subsystem is, for each assembly instruction specified in the task, a set of mating features which are viable, distinct (with respect to object symmetry), and ranked in order of plausibility. The highest ranked pair corresponding to each instruction is then hypothesized as the pair of mating features for that assembly operation. The FIND-FEATURES subsystem works by analyzing each assembly instruction in isolation.

Secondly, ASSEMBLE checks to ensure that the hypotheses generated by FIND-FEATURES for each of the instructions, when taken together and viewed as a potential solution to the specified assembly task, still prove viable. This process of consistency checking is carried out by the VERIFY subsystem. The VERIFY subsystem employs two kinds of reasoning for this consistency checking: reasoning about the accessibility of features, and reasoning about spatial relationships implied by the hypothesized mating features. Reasoning about the accessibility of features is carried out using the feature relationships that are a part of the object models and the specified precedence relationships between the assembly operations. When inconsistencies are detected, the ranked list of possible mating features produced by FIND-FEATURES is used to select alternatives and consistency checking is performed again. For reasoning about spatial relationships, VERIFY first generates the spatial relationships that form the input to the RAPT system, and then interfaces with RAPT to avail itself of RAPT's reasoning capability. The output of the VERIFY system is intended to be a sequence of feature-level descriptions of the assembly instructions, indicating a choice of mating features for each of the assembly operations that pass the accessibility and spatial relationship tests, and the spatial relationships implied by these choices for mating features. The implementations fall short of developing the interface from RAPT back to the VERIFY system.

The final step in the process is the generation of the output commands suitable for the special purpose planners ASSEMBLE is intended to interface with. This step is to be carried out by the GENERATE-MOVE-COMMANDS subsystem. This subsystem has not been implemented.

6.3 Delimitations

There are two dimensions along which the currently implemented system is limited. One is the set of assembly operations that ASSEMBLE can work with. The assembly operations that the present implementation handles are: INSERT, THREAD, PLACE, and DOUBLE-INSERT. These operations were chosen as they are some of the most commonly occurring assembly operations in practice [42]. For each of these operations, pertinent information such as the geometric constraints and spatial relations implied by the operation are encoded in the program. In order to extend the ASSEMBLE system to handle other operations, the above-mentioned information about these operations will have to be encoded into the system as well.

The other restriction on ASSEMBLE is the set of features of an object the system can represent and reason about. These are listed in Table 1. The choice of these feature types is not entirely independent of the choices made for the assembly operations. Rather, this set of features includes those that are likely mating features for the chosen assembly operations. An extension of the set of features considered will enhance the capabilities of the system. However, the purpose behind developing this system has, primarily, been as a means to evaluate the proposed object representation in terms of its effectiveness for reasoning about assembly and to evaluate the effectiveness of the reasoning mechanisms.

6.4 The Goal Network

The primary data structure used by ASSEMBLE to record the progress made in problem refinement is an AND/OR graph referred to as the *Goal Network*. The root node (at level 0) represents the entire assembly task. The nodes at level 1 represent

the given assembly instructions, and are referred to as *goals*. The different possible interpretations of these instructions specifying the mating features are represented as nodes at level 2. These are referred to as *feature-level subgoals* or *subgoals*. There are arcs that link the root node to each of the goals at level 1. Each of the level 1 nodes are linked to the feature-level subgoals at level 2 that represent its refinement. With this interpretation, the nodes at level 1 are AND nodes, and the nodes at level 2 are OR nodes. The relationship between the root node, goals, and subgoals defines a tree structure. However, the specified precedence relationships between the assembly instructions are represented as links between the goal nodes. Thus, it is more accurate to refer to this structure as a network rather than a tree. This network of goals and subgoals is altered as decisions are made during the process of assembly planning.

Setting up the goal network does not require that a total ordering of the goals be specified. The representation can accommodate any partial ordering of the goals. The ordering information is used only by the VERIFY subsystem in its accessibility analysis phase. The effectiveness of the accessibility analysis phase, however, is dependent upon the extent of ordering information.

6.4.1 Representation of Nodes

The nodes of the goal network are represented as structures. The information content of a node varies depending upon the level at which the node appears. The structure of a node at the object-level, in the network generated for the light-bulb fixture assembly, is shown in Figure 13. The values of the `parent-node` and `subgoals` slots of the node illustrate the structures of a root node and feature-level subgoal, respectively. Each node in the network contains a `description` of the task, goal or subgoal it represents. Also included are the `level` of the node and its type, namely, whether it is an AND or OR node.

The `parent-node` and `subgoals` slots provide pointers to the nodes to which a node is connected, at the levels above and below, respectively. The `predecessors` and `successors` slots are associated with goal nodes. They contain pointers to nodes at the same level to which a node is related through the specified precedence relations.

```
NODE-DESCRIPTION:          (insert part_b in part_a)
LEVEL:                     1
NODE-TYPE:                 and
SUBGOALS:                  (#S(subgoal-1
                           :node-description (insert ... )
                           :level 2
                           :node-type or
                           :confidence-measure 0.8333334
                           :parent-node #S#
                           :effects (# #)
                           :mating-feature-r a3
                           :mating-feature-i b1))
PARENT-NODE:               #S(root-node
                           :node-description "Light bulb
                                              Fixture Assembly"
                           :node-type and
                           :subgoals #)
PREDECESSORS:              nil
SUCCESSORS:                (#S#)
PRE-CONDITIONS:            (((accessible b1) (#S#))
                            ((accessible a3) (#S#)))
POST-CONDITIONS:           (((accessible a4) (#S#)))
CHOICE-POINTS:             (#S# #S#)
HEURISTICS-APPLIED:        (find-distinguishing-features
                                        closeness-of-fit)
RULES-APPLIED:             nil
```

Figure 13: Structure and Information Content of a Goal Node

The `preconditions` slot of a node specifies some of the conditions that have to be true in order that the operation represented by the node can be executed. The `effects` slot specifies conditions that are true as a result of executing the operation represented by the node. The `postconditions` slot specifies conditions that are required to be true after execution of the operation represented by the node.[2] Typically, the `postconditions` include `preconditions` of successor nodes that are likely to be affected by execution of the operation. `Preconditions` and `postconditions` are associated with nodes at level 1. `Effects` are associated with nodes at level 2. The `preconditions`, `effects` and `postconditions` slots are used in the accessibility analysis phase. Thus, the conditions that are of concern deal with the accessibility of features. The contents of these slots are further detailed in the description of the accessibility analysis phase of the VERIFY subsystem in Section 6.6.1.

The FIND-FEATURES subsystem determines, for each object-level goal, a set of feature-level subgoals which are viable, distinct with respect to object symmetry, and ranked in order of plausibility. The highest ranked subgoal is hypothesized as the interpretation of an object-level goal, and a pointer to this node is stored in its subgoals slot. The remaining feature-level subgoals are saved for later consideration by storing a list of pointers to these nodes in the `choice-points` slot. The `heuristics-applied` and `rules-applied` slots contain the list of heuristics and the list of rules that were applied to the goal represented by the node.

Each node at level 2 includes slots corresponding to the specification of mating features and a `confidence-measure` slot. The `confidence-measure` slot of any level 2 node holds a value between 0 and 1 which indicates its ranking among the subgoals of its parent level 1 node.

6.4.2 Initializing the Goal Network

When the input information, as specified in Section 6.1, is read in by ASSEMBLE, the system initializes the goal network by defining the root node, defining all of the level

[2]The use of the term "postconditions" here deviates from the traditional use of the term to denote effects of actions.

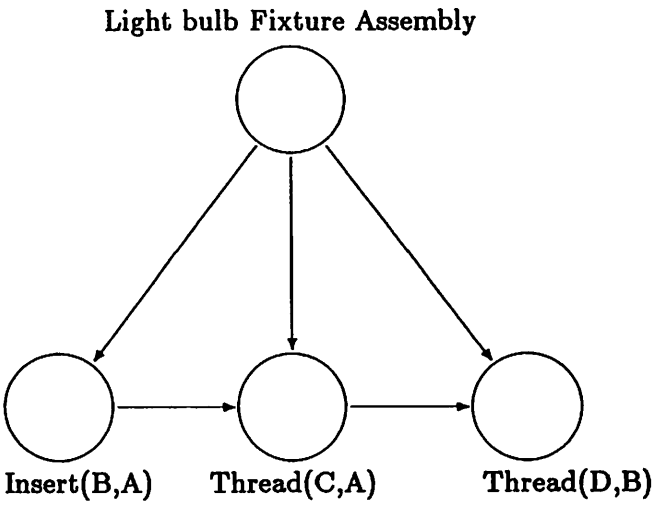


Figure 14: Initial Goal Network

1 nodes, establishing parent-child relationships, and establishing sibling relations as specified by the input precedence relationships. The root node is associated with the symbol `*goal-tree*`. The initial goal network corresponding to the light bulb fixture assembly which has been used for illustrative purposes in this chapter is shown in Figure 14. The initialized goal network is then processed by the FIND-FEATURES subsystem which expands the network to the next level by generating the feature-level subgoals and then restricts them in a manner to be described in the following section.

6.5 The FIND-FEATURES Subsystem

The function of the FIND-FEATURES subsystem is to identify the possible mating features corresponding to each of the input assembly instructions and rank them in order of plausibility. Given the initialized goal network, the subsystem generates the subtree emanating from each of the goal nodes, and then prunes the subtree, ranks the remaining nodes on the subtree, and selects the subgoal that serves as the interpretation of the goal node for further verification. The FIND-FEATURES subsystem processes the object-level goals one at a time, completing the growing and pruning processes for one goal

```
(defun find-features ()
  (mapcar 'find-features-aux (subgoals *goal-tree*)))

(defun find-features-aux (goal)
  (object-to-features goal)
  (use-beta-symmetry  goal)
  (use-alpha-symmetry goal)
  (initialize-conf-m goal)
  (if (> (length (subgoals goal)) 1)
      (apply-heuristics goal))
  (if (> (length (subgoals goal)) 1)
      (select-best-subgoal goal)))
```

Figure 15: Top-Level Functions of the FIND-FEATURES Subsystem

before processing the next (see Figure 15). The process of generating subtrees and the mechanisms used for pruning subtrees and ranking the subgoals are described in this section.

6.5.1 Generating Feature-Level Subgoals

In order to generate the feature-level subgoals of a goal, the system considers all possible combinations of features associated with the pair of objects specified in the goal, and passes these pairs of features through a filtering process. Geometric constraints associated with the assembly operation specified in the goal act as filter. Corresponding to each pair of mating features that passes the filter, a level-2 node is created and added to the tree as a child of the object-level goal.

While the operations INSERT, THREAD, and PLACE require specification of a single pair of mating features, the DOUBLE-INSERT operation requires specification of two pairs of mating features. The structure of a level-2 node falls into one of two classes depending on the assembly operation specified in the goal. The system only creates subgoals corresponding to pairs of mating features that satisfy the constraints imposed by the operation. Figure 16 depicts the goal network of Figure 14 after the level-2 nodes have been added.

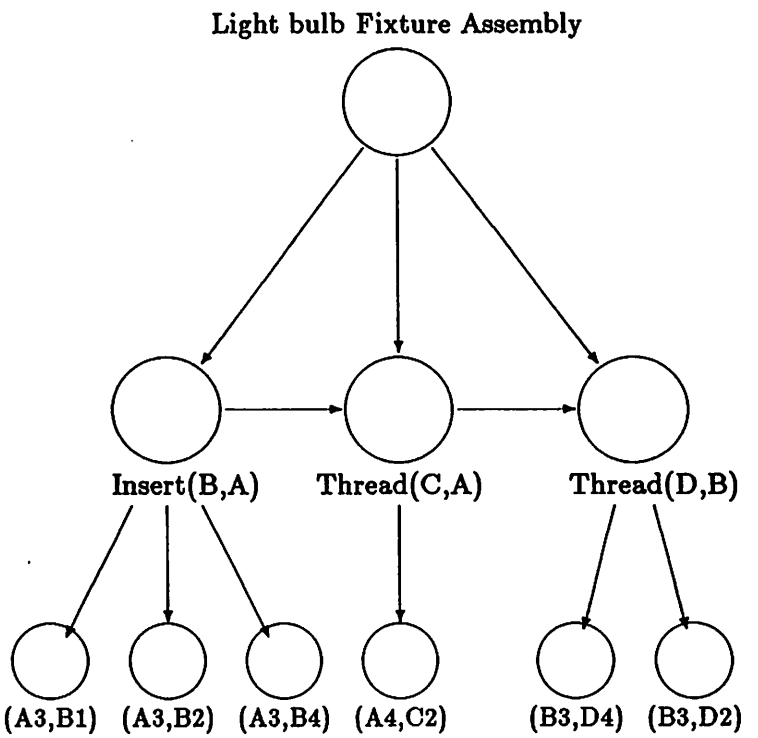


Figure 16: Goal Network with Level-2 Nodes

6.5.1.1 Operator-related constraints

The ASSEMBLE system includes constraints relating to the four operations the system is intended to handle: INSERT, THREAD, PLACE, and DOUBLE-INSERT. The constraints associated with each of these operations are listed below.

Constraints associated with the INSERT operation:

1. The dimensions of the mating feature of the receiver object should be at least as large as the mating feature of the insertor object.
2. The mating feature of the receiver object should be hollow.

Constraints associated with the THREAD operation:

3. The mating features of both objects should be of type 'thread'; The threading on the receiver object should be internal while the threading on the insertor object should be external.
4. The diameters of the cylindrical surfaces on which the threads lie should be nearly equal.
5. The pitch of the threads on either feature should be nearly equal.

Constraint associated with the PLACE operation:

6. Both mating features should be convex.

Constraints associated with the DOUBLE-INSERT operation:

7. The distance between the two mating features of the receiver object should be nearly equal to the distance between the two mating features of the insertor object.
8. Both features of the receiver object should be hollow.
9. The dimensions of the mating features of the receiver object should be at least as large as the corresponding mating features of the insertor object.
10. The relative orientation of the two receiver features (i.e., the angle between their normal axes) and the relative orientation of the insertor features should both be equal to zero.

Each of these constraints is encoded as a function and the list of functions corresponding to each operation is associated with the name of the operation in a property list. The

Table 3: Definition of the Dimension of Features

Shape	Size Parameters	Dimension
Cylindrical hole:	internal-diameter of the cylinder (d) depth of the hole	d
Rectangular hole:	length of the rectangle (l) width of the rectangle (w) depth of the hole	$\sqrt{l^2+w^2}$
Thread:	whether the thread is internal or external diameter (d) width of the thread pitch of the thread	d
Cylindrical shaft:	diameter of the cylinder (d) height of the cylinder	d
Rectangular plane face:	length of the rectangle (l) width of the rectangle (w)	$\sqrt{l^2+w^2}$
Circular plane face, Semi-circular face:	diameter of the circle (d)	d
Cylindrical surface, Semi-cylindrical surface:	height of the cylinder (h) diameter of the cylinder	h

functions that encode these geometric constraints operate on the feature attributes that constitute part of the input to the system. All of the constraints listed above with the exception of constraints 1, 7, 9, and 10 translate into simple functions that examine the values of the relevant feature attributes. Constraints 1 and 9 refer to the *dimensions* of the mating features. These constraints provide a simplified, although less accurate, version of the volumetric enclosure test. The *dimension* of a feature is defined as a linear metric that represents the length of the longest line contained in the surface of the feature, or the cross-section of the feature in cases where the feature is 3-dimensional entity such as a hole or a shaft. Table 3 provides the precise definition of *dimension* used in ASSEMBLE.

The relative orientation constraint, Constraint 10, is verified by computing the angles between the normal axes of the features as the arc cosine of the dotproduct of the unit vectors representing the axes. The distance constraint, Constraint 7, while being critical for the DOUBLE-INSERT operation, has proved difficult to realize computationally, based on the proposed object representation; the reason being that computing the distance

between centers of the two features is inadequate. Of relevance are the shortest and the largest distance between the periphery of the features. In the case of cylindrical shafts in cylindrical holes, the shortest distance between the periphery of the features can be computed by subtracting the radii of the holes (shafts) from the distance between the centers of the features. The largest distance is computed by adding the radii of the holes (shafts) to the distance between the centers of the features. In ASSEMBLE, the distance constraint is implemented in this fashion. However, this restricts the application of the constraint only to combinations of cylindrical and circular features.

Table 7 summarizes the results obtained from applying the constraints listed above for the operations in the five experimental assembly tasks on which ASSEMBLE has been used.

6.5.2 Pruning Subtrees using Object Symmetry

Once the level-2 nodes corresponding to a goal have been added to the goal network, the FIND-FEATURES subsystem attempts to reduce the number of subgoals of that goal by using the *α-equivalence* and *β-equivalence* relations defined on the set of features of an object (see page 53 for definitions of these relations). The procedure applied for the two types of symmetry are identical, and they are applied in sequence after the generation of level-2 nodes (see Figure 15). Described below is the procedure for detecting equivalent subgoals with respect to α-symmetry and β-symmetry. First, the system checks the values of the α-symmetry (β-symmetry) attributes of the two objects, say O_1 and O_2, specified in the goal. If the value of the α-symmetry (β-symmetry) attribute corresponding to either of the objects strictly lies between 0 and 2π, some of the subgoals may be equivalent and further processing occurs. Thus, the pruning procedure applies to a goal only when one or both of the objects involved in the goal exhibit finite symmetry.

If object O_1 has finite α-symmetry (β-symmetry), the system searches for pairs of subgoals with mating features (f_i, f_k) and (f_j, f_k) where:

- f_i and f_j are distinct features of O_1 with identical shape and size attributes;
- f_k is a feature of O_2;
- f_i and f_j are *α-equivalent* (*β-equivalent*) features of O_1.

For each such pair (f_i, f_k) and (f_j, f_k) identified, the subgoal corresponding to mating features (f_j, f_k) is deleted from the list of subgoals if the current goal is the only one in the task that uses object O_1. If O_1 is also used in other goals in the task, the subgoal corresponding to (f_j, f_k) is moved to the list of choice points. A similar search procedure is carried out with object O_2, if O_2 is α-symmetric (β-symmetric). The above algorithm applies to operations that involve a single pair of mating features, namely, INSERT, PLACE, and THREAD. The procedure is naturally extended for a DOUBLE-INSERT operation involving two pairs of mating features and is presented below.

Consider the case of a goal corresponding to a DOUBLE-INSERT operation involving objects O_1 and O_2. If object O_1 has finite α-symmetry (β-symmetry), the system searches for pairs of subgoals with mating features $((f_i, f_j), (f_k, f_l))$ and $((f_m, f_j), (f_n, f_l))$ where:

- f_i and f_m are distinct features of O_1 with identical shape and size attributes;
- f_k and f_n are distinct features of O_1 with identical shape and size attributes;
- f_j and f_l are distinct feature of O_2;
- f_i and f_m are *α-equivalent* (*β-equivalent*) features of O_1;
- f_k and f_n are *α-equivalent* (*β-equivalent*) features of O_1.

For each such pair $((f_i, f_j), (f_k, f_l))$ and $((f_m, f_j), (f_n, f_l))$ identified, the subgoal corresponding to mating features $((f_m, f_j), (f_n, f_l))$ is deleted from the list of subgoals if the current goal is the only one in the task that uses object O_1. If O_1 is also used in other goals in the task, the subgoal corresponding to $((f_m, f_j), (f_n, f_l))$ is moved to the list of choice points. A similar search procedure is also carried out with object O_2, when O_2 is α-symmetric (β-symmetric).

The inference that two features f_i and f_j of an object O_1 are *α-equivalent* (*β-equivalent*) is based on the representation of α-symmetry. Recall that representation of α-symmetry (β-symmetry) comprises specification of the axis of symmetry, k, with respect to the object coordinate frame, and the angle of symmetry, θ. The system computes the transformation T that corresponds to a rotation θ about k denoted by $Rot(k, \theta)$. The computation of the transformation T is based on the following general expression (see page 28 of [44]):

$Rot(k,\theta) =$

$$\begin{bmatrix} k_x k_x vers\theta + cos\theta & k_y k_x vers\theta - k_z sin\theta & k_z k_x vers\theta + k_y sin\theta & 0 \\ k_x k_y vers\theta + k_z sin\theta & k_y k_y vers\theta + cos\theta & k_z k_y vers\theta - k_x sin\theta & 0 \\ k_x k_z vers\theta - k_y sin\theta & k_y k_z vers\theta + k_x sin\theta & k_z k_z vers\theta + cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If a finite number of applications of T results in the center and normal-axis of f_i coinciding with those of f_j, then the features f_i and f_j are inferred to be *α-equivalent* (*β-equivalent*).

When there are finite symmetries in the objects specified in a goal, this strategy is fairly effective in reducing the search space. When the object that exhibits finite symmetry is not used in any other assembly operation, one of a pair of equivalent subgoals is deleted entirely from the search space. However, when the object is also used in other assembly operations, one of the pair of subgoals is moved to the list of choice-points as an alternative to consider at a later stage. Typically, if the angle of symmetry is θ, the reduction in the number of subgoals can be by upto a factor equal to $2\pi/\theta$. Table 7 summarises the results achieved by using object symmetry for the experimental assembly tasks run on ASSEMBLE.

6.5.3 Determining the Most Plausible Subgoal

As evidenced by the results presented in Table 7, the application of operator-related constraints and object symmetry alone are insufficient to uniquely determine the mating features for an operation, in most cases. Each feature-level subgoal of a goal node, whether in its sugoals list or its choice-points list, represents a possible interpretation of the goal. The FIND-FEATURES subsystem generates a hypothesis as to which one among these possible interpretations is most likely to be the correct interpretation. The hypothesis is then validated or invalidated by the analysis procedures of the VERIFY subsystem. The hypothesis generation is based upon:

(a) a set of heuristics relating to assembly that provides the basis of information for considering one set of subgoals of a goal node to be more plausible than the remaining subgoals;

(b) a numerical measure associated with each subgoal, referred to as its *confidence measure*, that represents the probability that the subgoal is the correct interpretation of its parent goal;

(c) a procedure that readjusts the confidence measures of subgoals based on the heuristic information.

The initial assignment of confidence measures to the set of feature-level subgoals of a goal node is uniform, as it is considered equally likely that any one of the subgoals included in the subgoals slot or choice points slot forms the correct interpretation of the goal node. The initial value of the confidence measure for each of the subgoals is given by 1 divided by the number of remaining subgoals of its parent node. The interpretation of the confidence measures as representing probabilities necessitates that any subsequent modifications to the confidence measures of the subgoals are effected so as to maintain the sum of the confidence measures of the subgoals of a goal node to be equal to 1.0.

6.5.3.1 Assembly Heuristics

The assembly heuristics do not provide conclusive information regarding the subgoals as the operator-related constraints and object symmetry information do. Instead, they embody some common-sense reasoning relating to assembly. The heuristics implemented are few in number, and represent an area in which the implementation can be strengthened further. The specific heuristics included in the implementation of the ASSEMBLE system are described below.

Closeness-of-fit: This heuristic is applicable to goals that represent insertion operations, that is, INSERT and DOUBLE-INSERT operations. Essentially this heuristic asserts that, if the pair of mating features corresponding to a subgoal represent as close a fit or a closer fit in comparison with the pair of mating features corresponding to any other subgoal, then the subgoal is a likely candidate for being the correct interpretation of its parent goal. The heuristic is related to the dimension constraints 1 and 9 of Section 6.5.1.1. The implementation of the heuristic, therefore, hinges on the definition of the *dimension* of a feature, provided in Table 3.

The algorithm that implements this heuristic works as follows: Let g denote a goal node such that:

(a) the assembly operation specified in g is one of the insertion operations;

(b) there is more than one feature-level subgoal corresponding to g.

Let $S = \{s_1, s_2, \ldots, s_n\}$ denote the set of subgoals of g. Let δ_i denote the difference in dimension between the mating features of subgoal s_i, for $i = 1, 2, \ldots, n$. The δ_is are computed, and their minimum determined as δ_{min}. The closeness-of-fit heuristic results in partitioning the set of subgoals into two sets P and D such that $P = \{s_i \mid \delta_i = \delta_{min}\}$ and $D = \{s_i \mid \delta_i > \delta_{min}\}$. The subgoals in P have their confidence measure value increased, and those of subgoals in D are appropriately decreased. The procedure used for modifying the confidence measures given sets P and D is described later in this section on page 86. No changes in the values of confidence measures result when $D = \emptyset$.

Table 8 summarises the results of applying the closeness-of-fit heuristic, and the modifications in the confidence rate measures for the insertion tasks occurring in the assembly experiments ASSEMBLE was exercised on.

Distinguishing features: This heuristic is applied to a goal node only when there is evidence to believe that exactly one its subgoals is the most plausible interpretation of the goal node, as reflected by the confidence measure values of the subgoals. The applicability condition can be stated as follows: Let g denote a goal node and let $S = \{s_1, s_2, \ldots, s_n\}$ denote the set of subgoals of g. Let c_i denote the confidence measure value of s_i for $i = 1, 2, \ldots, n$ and let $c_{max} = maximum(c_1, c_2, \ldots, c_n)$. Let $M = \{s_i \mid c_i = c_{max}\}$. The distinguishing feature heuristic is applicable to g if the cardinality of the set M equals 1. When the applicability condition is satisfied, let s_{max} denote the single element of M.

The main idea behind this heuristic is to obtain more detailed information about the objects involved in g in the vicinity of the mating features specified in s_{max}. Such information is assumed to be obtained by invoking the sensors, or by directing a query to the object database which in turn will invoke the sensors (see Figure 1). The additional information is then to be analyzed to gather further evidence in favor of the subgoal s_{max}. This heuristic was also chiefly motivated as a vehicle to test out our ideas on dynamic planning.

In ASSEMBLE, user-defined functions update the object information. This results in augmentation of the feature graphs, with new features and the relationships these

features hold with existing features of the objects. If no change results in the feature graph, the application of the heuristic does not alter the confidence measures of the subgoals. If indeed there are changes in the feature graphs of the two objects, the changes are compared and analyzed to see if there is additional evidence to increase the confidence measure of the feature-level subgoal. In this case too, as in the case of the closeness-of-fit heuristic, the set of subgoals S is partitioned into two subsets P and D, where $P = \{s_{max}\}$ and $D = S - \{s_{max}\}$. The procedure described in page 86 is then applied.

As the information required to update the object information is hand-coded by the user, and is specific to the objects in a particular task, and due to the difficulty in developing general principles as a basis for the analysis, the utility of this feature has been hard to substantiate. This heuristic was used in the light bulb fixture assembly of Figure 5. The application of this heuristic in that experiment is discussed in Section 7.4.

Functional Symmetry: This heuristic is not used for ranking subgoals, but to recognize subgoals that are functionally equivalent, in a manner similar to the *α-equivalence* and *β-equivalence* relations. The notion of *functional symmetry* is defined as follows: Two features from the same object are defined to be *functionally symmetric* if they are of the same type, have the same size parameters and bear the same relationships with other features of the object. The crucial difference between the *functional symmetry* relation and the *α-equivalence* and *β-equivalence* relations is that the former is independent of the position of the two features, while the latter, by definition, refer to positional symmetry.

The procedure used for application of this heuristic is fairly similar to the procedures used with α-symmetry and β-symmetry. The only applicability condition for this heuristic is that the goal node should have more than one feature-level subgoal in its subgoals slot. If g is a goal node that passes the applicability condition, the system searches for pairs of distinct subgoals s_1 and s_2 in its subgoals slot, with mating features (f_i, f_k) and (f_j, f_l), respectively, such that:

- f_i and f_j are functionally symmetric;
- f_k and f_l are functionally symmetric;

- $c_1 \geq c_2$ where c_i denotes the confidence measure associated with s_i, for $i = 1, 2$.

For each such pair of subgoals s_1 and s_2, the subgoal s_2 is moved to the list of choice points. Two features f_i and f_j are inferred to be functionally symmetric if either f_i and f_j are the same, or if distinct, they have the same shape and size attributes, and bear the same relationship with other features of the object. The extension of this procedure to goals representing operations that require more than one pair of mating features is identical to the procedure described for the application of object symmetry in Section 6.5.2 and is not repeated here.

Table 9 summarises the results of applying the functional symmetry heuristic on the experimental assembly tasks run on ASSEMBLE.

The *closeness-of-fit* and *distinguishing-features* heuristics rank the subgoals, while the *functional symmetry* heuristic partitions the set of subgoals into equivalence classes so that only a representative element from each equivalence class remains in the list of subgoals. The order in which these heuristics are applied in ASSEMBLE is the order in which they are presented above. The distinguishing-feature heuristic is the one heuristic that needs to be applied last among the ranking heuristics. Aside from this, the order in which the heuristics in the present set are applied can be arbitrary.

6.5.3.2 Procedure for Modifying Confidence Measures

Let g denote a goal node and $S = \{s_1, s_2, \ldots, s_n\}$ its set of subgoals. The heuristics that contribute towards ranking the subgoals define a partitioning of the set of subgoals S into two subsets P and D with the implication that the confidence measure values of the subgoals in P should be increased, and those of the subgoals in D should be decreased. Let ic_i denote the initial confidence measure of subgoal s_i $(i = 1, 2, \ldots, n)$ before modifications are effected. The confidence measure value of the subgoal s_i is modified to fc_i, for $i = 1, 2, \ldots, n$ where fc_i is computed as follows:

For each i such that $s_i \in D$,

$$fc_i = ic_i \div 2.$$

For each i such that $s_i \in P$,

$$fc_i = ic_i + \epsilon_i$$

where ϵ_i is determined as

$$\epsilon_i = ic_i(\sum_{j \in D}(fc_j - ic_j)/\sum_{k \in P} ic_k).$$

Table 8 illustrates the application of the modification procedure in relation to the closeness-of-fit heuristic.

The procedure used for modifying confidence measures in the ASSEMBLE system is fairly simple. The set of subgoals is partitioned into precisely two subsets and the treatment of the subgoals in each partition is uniform. In the case of the closeness-of-fit heuristic, the subgoals in D (the set of subgoals that have their confidence measures reduced) may exhibit a fairly wide range of differences in dimensions, and yet all of these subgoals are treated uniformly. Fairer and more complex schemes of redistribution can be devised. The simple procedure described above has been implemented in light of the fact that the collection of heuristics used in the current version of ASSEMBLE is small.

6.5.3.3 Hypothesis Selection

If, after application of the heuristics, there is more than one subgoal in the subgoals slot of a goal node, FIND-FEATURES selects among these the subgoal with the highest confidence measure as the most plausible interpretation of the goal node and moves the other subgoals to the list of choice-points for later consideration. When more than one subgoal satisfies this criterion, the first such subgoal that is encountered in the search process serves as the hypothesis for the goal node. The partial goal network corresponding to the light bulb fixture assembly task, where only the hypothesized subgoals are shown, is seen in Figure 17.

Choice-points, which represent alternate hypotheses, are generated by computing the partitions resulting from the equivalence relations defined by object symmetry and functional symmetry, and finally, by the hypothesis selection process.

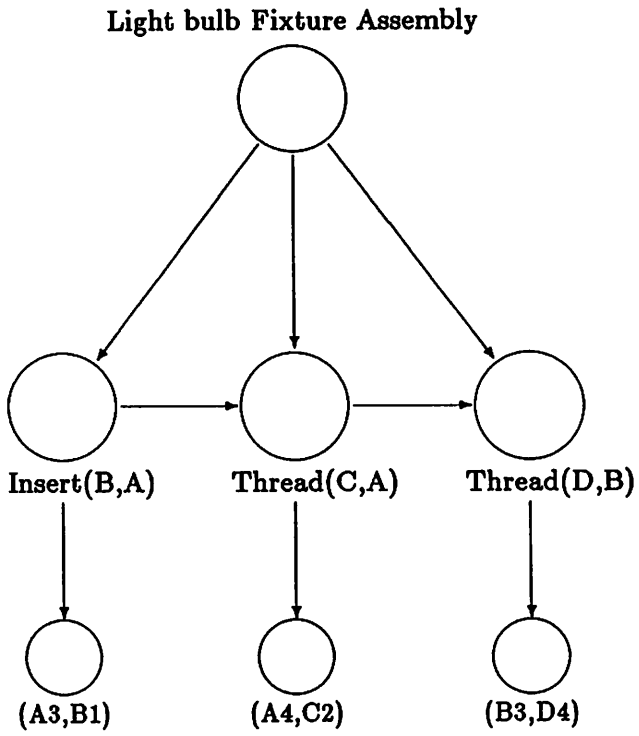


Figure 17: Goal Network with Hypothesized Subgoals

6.6 The VERIFY Subsystem

The function of the VERIFY subsystem is to evaluate the output of the FIND-FEATURES subsystem in an effort to determine if the hypotheses generated for the individual goal nodes are consistent and represent a coherent solution to the task, as a whole. The VERIFY subsystem employs two types of reasoning mechanisms for this evaluation. One is concerned with the accessibility of features, and the other is concerned with the spatial relationships implied by the hypothesized mating features and assembly operations specified. Of these, the accessibility analysis is carried out first, as it works with feature-level descriptions of the assembly instructions, represented as subgoals on the goal network. The spatial relationship analysis is carried out second, as this requires the generation of subfeatures of hypothesized mating features and the spatial relationships among these implied by the assembly operations. The procedures required to employ both of these analyses are described in this section.

6.6.1 Accessibility Analysis

The accessibility analysis works by the formulating, propagating and processing of constraints that specify the accessibility requirements for each assembly instruction (goal node). The accessibility analysis phase proceeds by generating the accessibility preconditions of all of the goal nodes, propagating these across the goal network, and analyzing each node for interactions. Each of these procedures is described below:

6.6.1.1 Constraint Formulation

The FIND-FEATURES subsystem leaves the hypothesized subgoal for each goal node in its subgoals slot and all other subgoals are placed in the choice-points slot. Corresponding to each goal node g of the goal network, and each mating features f_i specified in its hypothesized subgoal hs, the pair consisting of the predicate $(accessible\ f_i)$ and a pointer to the subgoal is included as a precondition to the goal node. Figure 13 illustrates the structure of a precondition.

6.6.1.2 Constraint Propagation

The accessibility constraints associated with each goal node are propagated forward to goal nodes that are its predecessors as postconditions in accordance with the following procedure:

Let g denote a goal node, and hs its hypothesized subgoal. Let $ac_g = ((accessible\ f_i)\ hs)$ be an accessibility constraint associated with g, and p a predecessor goal of g. Let F denote the set of mating features specified in the hypothesized subgoal of p. The accessibility constraint ac_g is associated with p as its post-condition if:

(a) $f_i \in F$; or

(b) $\exists f_j \in F$ such that f_i and f_j are features of the same object O and f_i and f_j are related by one of the relations defined between features in Section 4.4.

The postconditions associated with any goal represent the conditions that are required to be true after execution of the assembly operation specified in the goal. Thus, the precondition ac_g of one of its successor nodes g is associated with p, when it is likely that the achievement of p will have an effect on the truth or falsity of the predicate specified in ac_g. The accessibility preconditions and the postconditions resulting from the propagation phase for the light bulb fixture assembly are shown in Figure 18.

6.6.1.3 Interaction Analysis

Let $g_1, g_2, \ldots, g_n$ denote the set of goal nodes on the goal network. For each goal node g_i, $i = 1, 2, \ldots, n$, let $s_{ij}, j = 1, 2, \ldots, m_i$, denote the subgoals of g_i ranked by the values of their confidence measure slots. A pointer to the subgoal that is currently hypothesized as the interpretation of g_i, denoted hs_i, is stored in its subgoals slot and the remaining subgoals are accessible through its choice-points slot. Initially, s_{i1} is the hypothesized subgoal of g_i, for $i = 1, 2, \ldots, n$. Let $p_{ij}, j = 1, 2, \ldots, l_i$, denote the postconditions of g_i. The predicate $violates(hs, p)$ evaluates to *true* if the execution of the operation specified in g (parent node of hs) with the interpretation provided in hs results in p being *false*; it evaluates to *false*, otherwise. The objective of the interaction analysis phase is

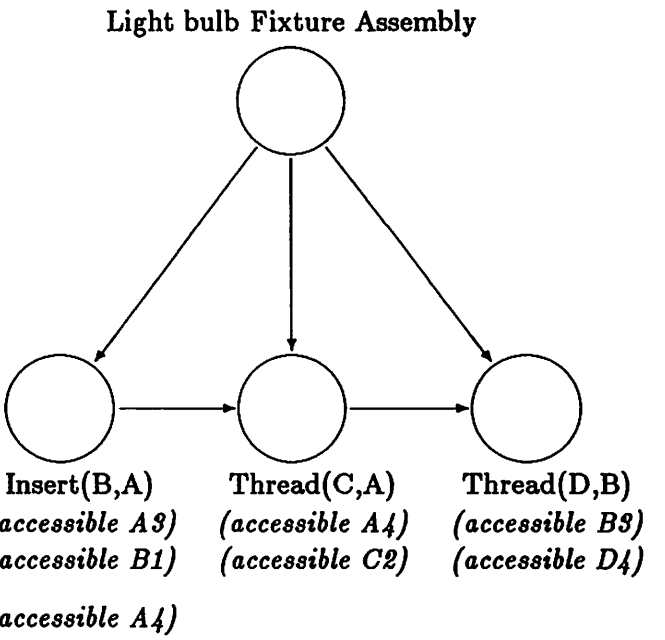


Figure 18: Goal Network with Accessibility Preconditions

to determine the hypothesized subgoal hs_i for each goal g_i, where $hs_i = s_{ij}$ for some $1 \leq j \leq m_i$, such that

$$\forall i \; \forall k \; \neg \; violates(hs_i, p_{ik}), i = 1, \ldots, n, \;\; k = 1, \ldots, l_i. \tag{6.1}$$

The computational procedure for the evaluation of $violates(hs, p)$, where hs is a feature level subgoal and p a postcondition of the form $((accessible\ f_i)(s_1\ \ldots\ s_k))$ consists of two parts. The first part of the procedure involves deducing the effects of executing hs in terms of accessibility of the mating features specified in hs. The second part involves evaluating the truth or falsity of p based on the deduced effects of hs.

Determining the effects of executing a feature-level subgoal hs is based on the assembly operation and the size attributes of the mating features specified in hs. Specifically, the procedure attempts to determine whether some of the mating features will be completely consumed by the execution of the assembly operation. This determination hinges on computing the area of the mating features specified in hs. As in the case of the computation of *dimension* (see Table 3), the area of cross-section is computed in cases where the feature is 3-dimensional, such as a hole or shaft. The precise interpretation of area as used in ASSEMBLE is provided in Table 4. In the case of an INSERT operation with mating features f_i and f_j, the area of the surfaces represented by features f_i and f_j are computed. If $area(f_i)$ and $area(f_j)$ are nearly equal, then it is concluded that neither feature will be accessible after the INSERT operation. The assertions $(delete\text{-}feature\ f_i)$ and $(delete\text{-}feature\ f_j)$ are added to the effects slot of hs. In the case of a PLACE operation with mating features f_i and f_j, the areas of the two features are computed. If $area(f_i)$ and $area(f_j)$ are nearly equal, both features are deduced to be inaccessible after the operation, and the corresponding assertions are added to the effects slot. If $area(f_i)$ is less than $area(f_j)$, then f_i is deduced to be inaccessible and the corresponding assertion added to the effects slot. The processing of the DOUBLE-INSERT operation is treated as two INSERT operations corresponding to the two pairs of mating features. In the case of a THREAD operation, both features are deduced to be inaccessible after the operation and the corresponding assertions are added to the effects of hs.

The second part of the procedure in determining the value of $violates(hs, p)$ where p is $((accessible\ f_i)(\ldots))$ involves checking to see if $(delete\text{-}feature\ f_i)$ is among the effects of hs. If so, the predicate $violates(hs, p)$ takes the value *true* and is *false* otherwise.

Table 4: Definition of the Area of Features

Shape	Size Parameters	Area
Cylindrical hole:	internal-diameter of the cylinder (d) depth of the hole	$\pi(d/2)^2$
Rectangular hole:	length of the rectangle (l) width of the rectangle (w) depth of the hole	lw
Cylindrical shaft:	diameter of the cylinder (d) height of the cylinder	$\pi(d/2)^2$
Rectangular plane face:	length of the rectangle (l) width of the rectangle (w)	lw
Circular plane face,	diameter of the circle (d)	$\pi(d/2)^2$
Semi-circular face:	diameter of the circle (d)	$\pi(d/2)^2/2$
Cylindrical surface, Semi-cylindrical surface:	height of the cylinder (h) diameter of the cylinder (d)	hd

The postconditions of a goal node are determined by the hypothesized subgoals of its successor nodes. Thus, changing the hypothesized subgoal of g_i will effect changes in the set of postconditions of the predecessors of g_i. For this reason, the goal nodes are arranged in a sequence $g_1, g_2, \ldots, g_n$ such that, if g_i is a successor of g_j then g_i appears earlier in the sequence than g_j. The interaction analysis phase processes each node in this sequence in succession.

For each goal node g_i, the processing consists of evaluating the truth value of

$$V_i = \bigvee_{k=1,\ldots,l_i} violates(hs_i, p_{ik}). \tag{6.2}$$

If V_i evaluates to false, then hs_i can serve as the hypothesized subgoal of g_i. However, if V_i evaluates to true, then at least one of postconditions of g_i is made false by the current choice of subgoal hs_i. This implies that the execution of the assembly opertion specified in g_i, with the mating features specified in hs_i, would make false a precondition of one of its successor nodes. In this case, an alternative subgoal from among the list of choice-points is chosen as the hypothesized subgoal hs_i involving the following steps:

- the pre-conditions of g_i and the post-conditions of its predecessors are based on the

value of hs_i that is no longer valid; thus, the preconditions of g_i as well as postconditions of predecessors of g_i that resulted from these preconditions are erased.

- the current value of hs_i is moved to the end of the list of choice-points.
- The subgoals in the choice-points slot are sorted in decreasing order of confidence-measure value. The earliest subgoal in the sequence for which the corresponding value of V_i evaluates to *false* is chosen as the current value of hs_i and is moved to the subgoals slot of g_i.
- The accessibility pre-conditions of g_i implied by the current value of hs_i are formulated.
- The accessibility constraints associated with the set of successor nodes of g_i are propagated forward to ensure that the postconditions of g_i reflect the current choice of hypothesized subgoals of its successor nodes.
- V_i is reevaluated. If V_i evaluates to *true*, the steps listed here are repeated; otherwise, the next goal node in the sequence g_{i+1} is processed. Note that determining the effects of hs_i is carried out only once for each time V_i is computed.

From the definition of V_i in (6.2), it is clear that the objective of the accessibility analysis phase expressed in (6.1) can be rewritten as

$$\bigwedge_{k=1,\ldots,n} \neg V_k. \tag{6.3}$$

The procedure described above ensures that at the time of processing g_{i+1},

$$\bigwedge_{k=1,\ldots,i} \neg V_k$$

is true, incrementally achieving the desired goal expressed in (6.3). Thus, processing the goals nodes in the prearranged sequence guarantees that when all goal nodes are processed, the desired result expressed in (6.1) is accomplished.

The effectiveness of the algorithm described above hinges on the order in which the goal nodes are presented. The goal nodes are processed in the reverse order of their execution. This approach is referred to in the planning literature as *goal regression* [62]. An alternative would be to process the goal nodes in the order in which they are intended to be executed. In this case, instead of propagating the preconditions forward to preceding nodes on the goal network, effects of operations will be propagated to succeeding nodes

on the goal network. The former approach was adopted as it appeared, on the surface, that there would be a smaller likelihood of encountering situations where backtracking is required. However, a systematic evaluation of the two modes of control is required to determine the relative efficiency of the approaches. None of the experiments on which ASSEMBLE was exercised required backtracking. The results of applying the accessibility analysis on each of these assembly tasks are described in Chapter 7. The procedure was found to be effective in identifying the use of the same feature on different operations and finding alternative subgoals that did not yield such conflicts.

An effective implementation of the $violates(hs, p)$ predicate is a crucial factor in the accessibility analysis phase. Both components of the procedure that implement this predicate can be further refined, and this provides an area in which this research can be extended. In the first part of the procedure, the effects of operations are represented abstractly as (*delete-feature* f_i). This does not adequately represent situations where a feature is partially consumed such as the insertion of a half-cylinder in a cylindrical hole. An example of such a situation is found in Figure 23. Consequently, the second part which comprises the evaluation of the truth or falsity of p, does not take into consideration the subsequent assembly operation which caused p to be a post-condition. An investigation of the use of solid modelling systems, which provide a more precise volumetric representation of the objects, to determine their usefulness in characterizing the effects of assembly operations as well as the in evaluation of postconditions given the simulated effects of the current choice of mating features, is planned as a future extension of this work.

While the accessibility analysis phase is successfully completed when the expression in (6.1) evaluates to true, the truth value of the expression does not guarantee that the assembly operations with their corresponding interpretations can be carried out simultaneously. There are two ways in which the proposed assembly may prove infeasible:

- There could be partial overlap of the volumes of objects, which the accessibility analysis phase does not address.
- The accessibility analysis is also insensitive to relative positions of features in objects and resulting subassemblies. Thus, the accessibility anaysis can generate hypotheses that are spatially inconsistent. An example of this situation occurs in the NC-100 controller assembly discussed in Section 7.5.

6.6.2 Reasoning about Spatial Relationships

In this phase, the VERIFY system derives the spatial relationships implied by executing the assembly operations using the mating features specified in the hypothesized subgoals. This involves generating the subfeatures corresponding to these mating features and determining locations of the subfeatures with respect to the object coordinate frames. The subfeatures generated are also features in RAPT parlance. The spatial relationships are specified as existing between the subfeatures. The ideas underlying the derivation of spatial relationships and a proposed set of spatial relationship predicates are detailed in Section 5.2. The implementation restricts itself to the use of the *fits* and *against* relations. These two relations adequately capture the implied spatial relationships for the set of assembly operations and the restricted class of feature geometry considered in this study. The type and location of these subfeatures as well as the spatial relationships are then processed by RAPT which performs the spatial reasoning to determine whether the spatial relationships arising from the various goals can be simultaneously satisfied. The remainder of this subsection describes the procedure for generating the subfeatures and information pertaining to them, generating the spatial relationships between the subfeatures, interface with the RAPT system, the input information required, and the procedure by which RAPT input is generated.

6.6.2.1 Generating Subfeatures

The notion of subfeatures as representing the constituent surfaces of a feature has been introduced in page 43 of Chapter 4. All of the reasoning carried out in the ASSEMBLE system thus far is based on feature attributes. However, in the spatial relationship analysis the reasoning is based on information about subfeatures. Each subfeature generated can be viewed as a RAPT feature, and this facilitates utilizing RAPT for spatial reasoning. Table 5 lists the subfeatures generated corresponding to each of the feature types handled by ASSEMBLE and its type as a RAPT feature.

The following observations serve to explain the list of subfeatures corresponding to each type of feature (for not all subfeatures correspond to physical surfaces of a feature), as well as the relevance of the assembly operation in determining what subfeatures are

Table 5: Subfeatures corresponding to the different feature types

Assembly Operation	Shape of Feature f	ASSEMBLE subfeature	Type of RAPT feature
All	cylindrical hole	f_curved_face f_bottom_face	cylindrical_face plane_face
	rectangular hole	f_bottom_face f_front f_right f_back f_left	plane_face plane_face plane_face plane_face plane_face
	thread	f_curved_face f_bottom_face	cylindrical_face plane_face
	cylindrical shaft	f_curved_face f_bottom_face	cylindrical_face plane_face
Insert, Double-insert	rectangular face, circular face, semi-circular face, cylindrical surface, semi-cylindrical surface	f_curved_face f_bottom_face	cylindrical_face plane_face
Place	rectangular face, circular face, semi-circular face, cylindrical surface, semi-cylindrical surface	f_plane	plane_face

generated.

1. For the purposes of this analysis using spatial relations, the THREAD operations are interpreted as if they are INSERT operations involving cylindrical holes and cylindrical shafts where the extent of insertion is controlled by the width of the threads on the two mating features. Thus, if f is a feature of type *thread*, the subfeature f_bottom_face generated corresponds to a virtual surface instead of a physical surface that is part of the feature f.

2. If, as part of the specified assembly, a non-cylindrical object is to be inserted into a cylindrical hole (object B of the light bulb fixture assembly of Figure 5 provides an example), the mating feature is often determined as a surface (e.g. feature B1 of object B) rather than as a volume. However, volumetric containment is implied by an INSERT operation. Therefore, the convex mating feature feature f, corresponding to an INSERT operation and a concave mating feature which is cylindrical, is always viewed as a cylindrical shaft. The center and normal axis of the shaft coincide with those of f and the diameter of the cylindrical shaft is determined as the *dimension* of feature f as defined in Table 3. Thus, f_curved_face corresponds to a virtual cylindrical surface.

Subfeatures are generated for each of the mating features corresponding to the hypothesized feature-level subgoals on the goal network. This involves instantiating a structure template associated with subfeatures and filling in the information pertaining to the subfeature. The following information is associated with each subfeature: name, type (as a RAPT feature), shape, dimensions and position. As with feature attributes, the attribute's name, type and shape are represented as symbols, dimension as a list of numbers, and position as a 2×4 array encoding the center and normal axis associated with the subfeature. The structures corresponding to the subfeatures of feature A3, a cylindrical hole, seen in Figure 5 are shown in Figure 19. Attributes of feature A3 are shown in Figure 11.

The position of each subfeature, that is, its center and normal axis, are the same as its parent feature except in the case of the subfeatures of a cylindrical hole. For a feature f which is a cylindrical hole of depth d and whose position is specified by its center (c_x, c_y, c_z) and a unit vector in the direction of its normal axis (n_x, n_y, n_z), the positions of its subfeatures f_curved_face and f_bottom_face are given by the tuple $(\mathbf{c}, \mathbf{n})$ where

$$\mathbf{c} = (c_x, c_y, c_z) - d * (n_x, n_y, n_z)$$

```
SUBFEATURE-NAME:          a3_curved_face
SUBFEATURE-TYPE:          cylindrical_face
SUBFEATURE-SHAPE:         cylindrical-surface
SUBFEATURE-DIMENSIONS:    (3.6 2.0)
SUBFEATURE-POSITION:      #2A((0.0 0.0 -0.7999 1.0) (0 0 1 0))

SUBFEATURE-NAME:          a3_botttom_face
SUBFEATURE-TYPE:          plane_face
SUBFEATURE-SHAPE:         circular-face
SUBFEATURE-DIMENSIONS:    (3.6)
SUBFEATURE-POSITION:      #2A((0.0 0.0 -0.7999 1.0) (0 0 1 0))
```

Figure 19: Subfeature Attributes

$$\mathbf{n} = (n_x, n_y, n_z).$$

The center is translated along the axis opposed to the normal axis in the case of a feature that represents an internal thread as well. As mentioned before, the operation THREAD is treated as an INSERT. In this case, the extent of translation d is determined as the minimum of the widths of the two threaded features.

6.6.2.2 Generating Spatial Relationships among Subfeatures

Section 5.2 provides an enumeration of the spatial relationship constraints implied by each of the assembly operations considered in this study. The earlier discussion expresses the spatial constraints in terms of the position attributes of features and contact relationships between feature surfaces. Having precisely defined subfeatures and their positions, it is now feasible to express the same spatial constraints in terms of subfeatures and their positions. This allows us to eliminate some of the resulting redundancy in the constraints.

The constraints on page 56 corresponding to an INSERT operation that is a *close-fit* between features $feature_k$ and $feature_l$ can be restated in terms of their subfeatures as follows:

1. opposed((x_k, y_k, z_k), (x_l, y_l, z_l))

2. against($feature_k$_bottom_face,$feature_l$_bottom_face)

Table 6: RAPT relationships generated for each assembly operation

Assembly operation	Mating features	RAPT spatial relationship generated
Insert	f_1, f_2	f_1_curved_face *fits* f_2_curved_face f_1_bottom_face *against* f_2_bottom_face
Double-insert	f_{11}, f_{12}	f_{11}_curved_face *fits* f_{12}_curved_face f_{11}_bottom_face *against* f_{12}_bottom_face
	f_{21}, f_{22}	f_{21}_curved_face *fits* f_{22}_curved_face f_{21}_bottom_face *against* f_{22}_bottom_face
Thread	f_1, f_2	f_1_curved_face *fits* f_2_curved_face f_1_bottom_face *against* f_2_bottom_face
Place	f_1, f_2	f_1_plane *against* f_2_plane

3. fits($feature_k$_curved_face,$feature_l$_curved_face)

4. $center_{k\_bottom\_face} = center_{l\_bottom\_face}$

where (x_k, y_k, z_k) represents the normal axis of $feature_k$ and its subfeatures, and similarly (x_l, y_l, z_l) represents the normal axis of $feature_l$ and its subfeatures. Of the spatial realtionships listed above, relationship 1 is implied by relationship 3 and relationships 2 and 3 together imply relationship 4. Similarly the constraints associated with the PLACE operation between $feature_k$ and $feature_l$ can be collapsed into a single relationship:

against($feature_k$_plane, $feature_l$_plane).

As noted earlier, the THREAD and DOUBLE-INSERT operations are treated as special cases of INSERT and are not discussed separately. Table 6 lists the RAPT spatial commands generated by ASSEMBLE for each of the assembly operations.

6.6.2.3 Interface with RAPT

Currently, the interface with the RAPT system is through a limited version of the Edinburgh Designer System (EDS). EDS [48] is a system for engineering design comprising many modules. The EDS system is implemented in the Poplog programming environment which includes Prolog, Pop-11 and Common Lisp. The primary reason for

using EDS as a front end for communicating with RAPT is the ease of interface. Of the several components of EDS, the taxonomy of module definitions, Assumption Based Truth Maintenance System (ATMS), and Spatial Relationship Engine (SRE) are the ones directly involved in the interface with the VERIFY system. The primary component involved in the reasoning process is the SRE which is the RAPT system described in Section 5.3 interfaced with the ATMS. The set of module definitions used in the interface are very small comprising only the definitions of feature types known to RAPT such as *cylindrical-face* and *plane-face*. The ATMS performs housekeeping by keeping track of consistent sets of assumptions.

For the purposes of the work detailed here, it is only important to know that the information required for processing by RAPT is input to the EDS as a set of assumptions, maintained by the ATMS. The SRE works with this set of assumptions and attempts to derive consequences either as tighter spatial relationships or, where possible, locations of objects. It also reports any inconsistencies found in the set of assumptions.

RAPT requires the following input information concerning an assembly in order to preform spatial reasoning:

- An identification of a set of RAPT features.
- For each RAPT feature identified,
  - assembly object to which it belongs;
  - type of the feature;
  - relevant parameters that define the feature; e.g. the radius of the cylinder when a feature is declared to be of type *cylindrical-face*;
  - location of the feature with respect to a coordinate frame embedded in the object on which the feature resides. The locations of features are specified as coordinate frames. Figure 20 illustrates the conventions adopted by RAPT in defining feature locations.
- A set of spatial relationships that are known to exist between these RAPT features.
- The absolute location of one of the objects involved in the assembly.

The set of subfeatures generated constitute the set of RAPT features. Most of the required information about RAPT features, such as type, parameter and parent object are available in the subfeature attributes or are easily obtained. The feature location

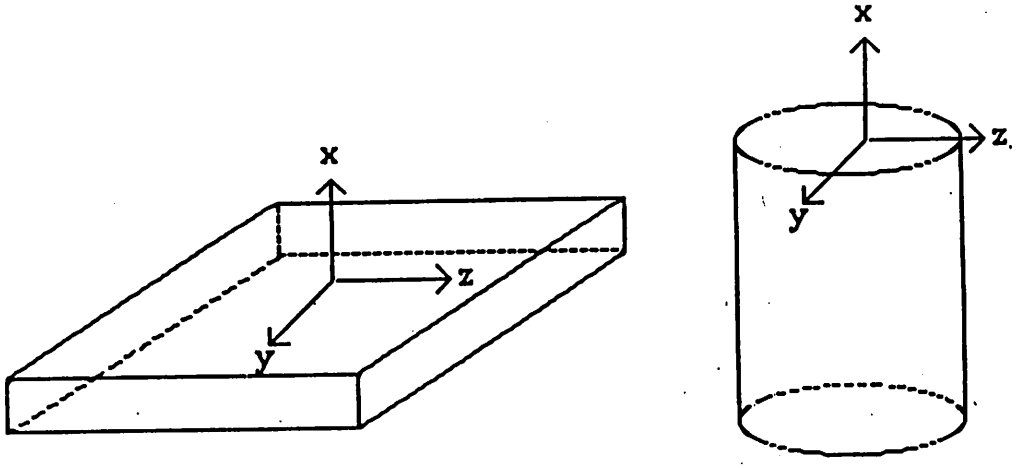


Figure 20: Conventions adopted for specifying locations of features in RAPT

required by RAPT, however, is different in structure and information content from the position attribute of subfeatures. RAPT requires that a right-handed coordinate frame be associated with each feature with the convention that the X-axis of the frame be pointing away from the object. Thus, the X-axis corresponds to the normal axis of a subfeature (see Figure 20). The origin of the coordinate frame coincides with the center of the subfeature. Additional constraints are required to determine the direction of the Y-axis of the frame. The system chooses the Y-axis direction arbitrarily so as to be perpendicular to the X-axis, and the Z-axis is then computed as the cross-product of the X and Y-axes. The spatial relationships are generated as per Table 6. Choice of that object which will serve as the reference and relative to which positions of other objects will be computed is obtained from the user.

A Prolog file encoding the above information as a set of assumptions is generated. The file generated corresponding to the light bulb fixture assembly of Figure 5 is shown in Section D.3 of Appendix D. The set of assumptions are then processed by SRE resulting in the identification of relative locations of objects or conflicts in the set of assumptions. Examples of both scenarios are shown in the next chapter which discusses the behavior of ASSEMBLE on a few assembly tasks.

6.7 Summary

This chapter has provided a description of a working system that couples the object representation and reasoning processes outlined in the previous two chapters and produces hypotheses about the resulting assembly configuration. The basic tenet underlying the object representation as well as the hierarchical decomposition approach to reasoning is that it is computationally more efficient to perform some informal analysis to generate a good set of hypotheses and then to identify an acceptable solution by using formal analyses as a process of verification. The necessary components for a system that attempts to reason about assembly are:

- an ability to represent objects volumetrically and reason about the interaction of these volumes in the realization of the assembly;
- an ability to represent locations of different features of an object and reason about spatial consistancy in the realization of the assembly;
- an ability to reason about the reachability of features in a partial assembly, since, even when an assembly is feasible in terms of volume availability and spatial consistency, the feature may be occluded so that it would be impossible to realize the assembly.

Of these, spatial reasoning has been effectively integrated into the work described here. Limited analysis of volumetric interference is a part of the application of operator related constraints as well as the accessibility analysis phase. The work described here can be usefully extended by interfacing with a solid modelling system such as PADL-2 [12] or ROBMOD [14] and utilizing its ability to detect volumetric interference. The reachability analysis is a less tractable problem. However, a reasonable solution to the problem is critical for actualization of the assembly.

An intended extension of the system is the integration with motion and manipulation planners as indicated in Section 5.4. Typically, motion planning systems address the problem of generating trajectories for the controllable joints of a manipulator that will result in moving the manipulator end point from a specified start location to a specified goal location while avoiding collisions with objects in the environment. Assembly involves devising motion strategies that would effectively make a choice as to the order in which

the implied spatial relationships between subfeatures will be accomplished. The choices will depend on the geometry of the features as well as the surrounding environment. It would make an interesting study to attempt to understand the precise nature of the constraints that lead to the choice of motion strategies.

On the discussion of the interface between the ASSEMBLE and RAPT systems, it was observed that the two systems have different ways for representing the locations of features. The implication of these different conventions is that, by using a single axis to define the location of a feature ASSEMBLE assumes rotational symmetry of the feature about its normal axis. This causes ASSEMBLE to run into problems when dealing with situations such as inserting a rectangular peg in a rectangular hole. RAPT, on the other hand, ignores any symmetry in the feature by associating a coordinate frame with each feature. Thus, presented with the situation where a cylindrical peg is being inserted into a cylindrical hole, RAPT identifies a precise location of the peg with respect to the hole, ignoring the rotational symmetry that exists in their spatial relationship. The rotational symmetry, however, is valuable information for a manipulation system attempting to carry out the assembly. An effective compromise would be to use a coordinate frame with each feature and include the representation of feature symmetry as a feature attribute and use this in the spatial reasoning process. Explicit representation of symmetry and related reasoning will enhance the capabilities of RAPT causing it to not overconstrain the problem in ways that result in loss of valuable information.

CHAPTER 7

EXPERIMENTS

This chapter discusses the experiments conducted using ASSEMBLE to evaluate the usefulness of the representation and reasoning strategies described in the earlier chapters. A set of five different assembly tasks of varying complexity have been used in the experiments. The work described in this dissertation evolved from studying very simple assembly tasks, each of which focused on one or two aspects pertinent to assembly. They involve few objects, typically three or four. The assembly of an NC-100 controller involving 14 different objects, and 26 assembly operations that relate these objects, has been modeled and processed by ASSEMBLE. The performance of ASSEMBLE on four of the simple assembly tasks, which serve to illustrate the usefulness and limitations of the reasoning strategies used, and the more complex NC-100 controller assembly are described in this chapter. The first three experiments are conducted on synthetic data while the last two experiments use real data derived from using the object representation techniques described earlier in Chapter 4 to model physical objects.

An assembly can be represented as a graph in which each node represents an object, and each arc represents an assembly operation that relates the objects represented by the two nodes connected by the arc. The *complexity* of an assembly can be then be measured as the *nullity* or *cyclomatic number* of the corresponding graph representation. For a graph G with n nodes, e arcs, and k components, the *nullity of* G, denoted by N, is given as

$$N = e - n + k. \tag{7.1}$$

The graphs corresponding to the five assembly tasks discussed in the following sections are shown in Figures 22, 24, 26, 28, and 30, respectively. The first four assembly tasks discussed have a complexity measure of 0, and the final assembly task has a complexity

measure of 14. The computation of complexity of an assembly is based on representation of the assembly as an undirected graph. However, the assembly operations represented by the arcs do not define a symmetric relation between the objects. Thus, the arcs in the graphs shown in Figures 22, 24, 26, 28, and 30, are directed so as to indicate the roles played by the objects represented by the nodes in the assembly operation represented by the arc.

The first experiment discussed requires the insertion of two identical cylindrical pegs in a circular plate containing two cylindrical holes. The objects and their features are shown in Figure 21. The holes are symmetrically placed on the circular plate. The cylindrical pegs have infinite symmetry about the axis of the cylinder and finite symmetry about axes perpendicular to the axis of the cylinder. The objects are simple and are easily represented. This experiment illustrates the usefulness of the representation of object symmetry, and the techniques by which this information is applied, in reducing the search space. It also demonstrates the use of the accessibility analysis procedure in detecting interactions between solutions to subproblems, and arriving at a valid final assembly configuration.

The second experiment requires the insertion of two semi-cylindrical pegs that should fit snugly into a cylindrical hole in a third object. The objects and their features are shown in Figure 23. This assembly task has been taken from [29]. Unlike the previous experiment where two different features of an object are involved in the two insertion tasks, this assembly requires the same feature of an object, the cylindrical hole, to be the receiver in both insertion tasks. Volumetric interference analysis becomes critical in this case, and as the account in Section 7.2 shows, the locations deduced by RAPT for both semi-cylindrical pegs result in coincident volumes. The desired outcome, however, is to have the volumes of the semi-cylindrical pegs be non-interfering.

The third experiment focuses on DOUBLE-INSERT operations. One of the objects involved in the task is a rectangular block having six holes. Of these, one hole is larger, while the other five are of the same diameter. The five smaller holes are placed in such a way that their distance from the larger hole is the same. Four of the five smaller holes are also placed in a sequence such that the distance between any two holes adjacent in the sequence is the same. There are three objects each of which have two protruding

cylindrical shafts. One of the objects has shafts of unequal diameters. One of its shafts has the same diameter as the larger hole and the other has the same diameter as the smaller holes on the rectangular block. The other two objects are identical, and are both symmetrical having shafts of the same diameter as the smaller holes on the rectangular block. The three objects are to be inserted into the rectangular block. This example has appeared before in [59]. The objects and their features are shown in Figure 25. DOUBLE-INSERT operations provide interesting spatial constraints, which are encoded as distance constraint and relative orientation constraint in ASSEMBLE as described in page 78. Aside from serving to validate the procedures used for DOUBLE-INSERT operations, this experiment demonstrates the use of object symmetry, and the closeness-of-fit heuristic and associated procedures for ranking subgoals. It also demonstrates the usefulness of the notion of functional symmetry, and the accessibility analysis procedures for detecting and correcting interactions between subgoals.

The fourth experiment is the assembly of the light bulb fixture which was introduced in Chapter 4 and used throughout Chapter 6 for illustrative purposes. Figure 27 shows the components of the assembly. The objects have complex shapes and include primary features which are the predominant features relevant for assembly, and secondary features which can provide additional constraints for the assembly. The objects with an identification of their primary features is shown in Figure 5. Note the flanges and grooves on objects A and B and the obtruding lip on object D which constitute secondary features. The assembly involves an INSERT operation and two SCREW operations. This experiment uses the *closeness-of-fit* and *distinguishing-features* heuristics and demonstrates the usefulness of the mechanism for ranking subgoals. However, ASSEMBLE only achieves partial success with this experiment. While the positions of objects B and C are deduced correctly relative to object A, the position of object D is deduced incorrectly. At the time of assembling object D with object B, object C is affixed as a sleeve around B. So, unless the end of object D with the obtruding lip is used, the screwing operation can only be partially completed. An ability to represent the volumes of objects and using associated reasoning can detect this failure.

The last experiment discussed in this chapter is the complex assembly of an NC-100 controller. The assembly task was used for studying grasping and manipulation in [37]

and has been adopted here with a few minor modifications. The assembly consists of a base-plate, a transformer, a capacitor, two side-plates, a top-plate, four large bolts, and four small bolts. The components of the assembly are shown in Figure 29. The transformer is placed on the base-plate and affixed to it using the four small bolts. The capacitor is inserted into a hole on the base-plate. The two side-plates are placed on the base-plate, the top-plate is placed on top of the side-plates, and all four objects are affixed together using the four large bolts. The transformer and the side plates are symmetrical objects. In addition, there are several sets of objects within which objects can be used interchangeably, such as the small bolts, large bolts and side plates. This situation leads to a large search space, but is reduced effectively by the use of functional symmetry and corrections applied by the accessibility analysis phase. More importantly, this experiment demonstrates the usefulness of the spatial reasoning component. In the case of fastening the transformer to the base-plate using a small bolt, the fastening operation is represented as two assembly operations in the set of input instructions to ASSEMBLE: an insertion of the bolt into a hole in the transformer and a screwing operation of the bolt into the base-plate. This results in *fits* relationships between the small bolt and the hole in the transformer, and between the small bolt and the hole in the base-plate, as shown in Table 6. From this RAPT deduces that the holes on the base-plate and transformer should be aligned. Recall the analogous box, lid and bolt scenario discussed in page 61 of Section 5.3. Similarly, alignment relationships between the base-plate, side-plate and top-plate are deduced. However, as the discussion of the NC-100 controller assembly in Section 7.5 shows, the alignments suggested by the hypothesized subgoals at the end of the accessibility analysis phase are spatially inconsistent in both the base-plate–transformer affixment as well as the base-plate–side-plate affixment. RAPT detects these inconsistencies in both cases.

7.1 Experiment 1: Pegs in Holes Assembly

The objects in the assembly and the features of these objects that were modeled are shown in Figure 21. The graph corresponding to the assembly indicating its complexity is shown in Figure 22. The circular plate, part_e, is modeled as having three features:

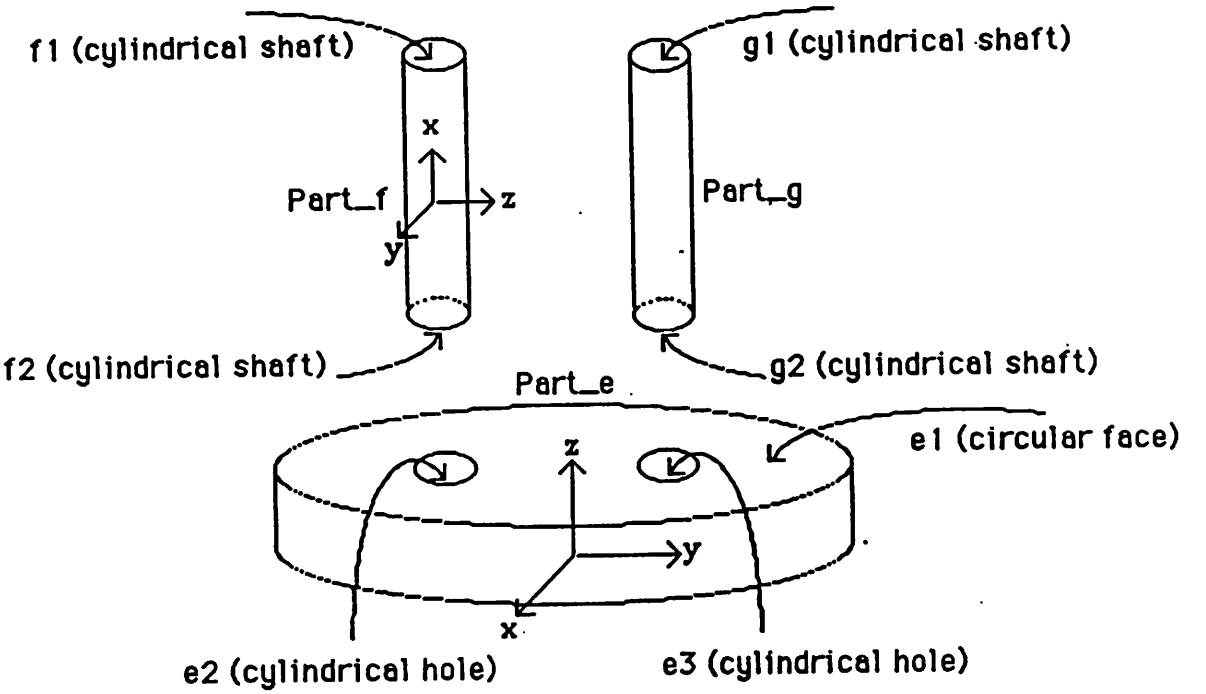


Figure 21: Peg Insertions

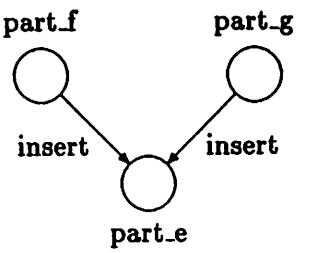


Figure 22: Assembly graph for Pegs in Holes Assembly

a circular face (e1), and two cylindrical holes (e2 and e3). The pegs, part_f and part_g, are modeled as having two overlapping cylindrical shafts, f1 and f2, and, g1 and g2, as features. Each peg is modeled as two cylindrical shafts so as to be able to capture the fact that either end of the peg can be used for insertion. The symmetry attributes of these objects as represented in the object models are shown below:

Object	α-symmetry	β-symmetry
part_e	2π	π
part_f	π	0
part_g	π	0

The structures corresponding to the objects and their features, that constitute part of the input information to ASSEMBLE, are shown in Section A.1.1. As is seen from the symmetry attributes of these objects, part_e is modeled as having β-symmetry about its Z-axis, and part_f and part_g are modeled as having α-symmetry about their respective Y-axes.

The assembly instructions given as input to ASSEMBLE are:

(G-1): `Insert part_f into part_e`

(G-2): `Insert part_g into part_e`

(G-3): `(G-1) precedes (G-2).`

The FIND-FEATURES subsystem processes each of these goals in succession. As the assembly operation and the objects involved are identical for both goals, the processing of one of the goals, (G-1), by this subsystem is presented here in detail, and the results obtained for the other goal are presented. A complete trace produced by ASSEMBLE for this experiment is provided in Section A.2.

Application of the geometric constraints associated with the INSERT operation results in the generation of the following feature-level subgoals corresponding to goal (G-1):

(sg1-1): `Insert feature f2 of part_f into feature e3 of part_e`

(sg1-2): `Insert feature f2 of part_f into feature e2 of part_e`

(sg1–3): `Insert feature f1 of part_f into feature e3 of part_e`

(sg1–4): `Insert feature f1 of part_f into feature e2 of part_e`

Part_e is β-symmetric. Applying the procedure described in Section 6.5.2, FIND-FEATURES finds subgoals (sg1-1) and (sg1-2) which specify the same feature, f2, as the mating feature from part_f and features e2 and e3 as mating features from part_e. It then verifies that features e2 and e3 are β-equivalent. Part_e is involved in other operations in the assembly; therefore, the subgoal (sg1-2) is moved to the choice-points slot. Subgoals (sg1-3) and (sg1-4) are in a similar relationship to each other, so the procedure also results in moving (sg1-4) to the choice-points slot. The status of the subtree of the goal network rooted at (G-1) is summarized as follows:

```
Feature Level Subgoals

(insert feature f2 of part_f into feature e3 of part_e).

(insert feature f1 of part_f into feature e3 of part_e).


Feature Level Subgoals deferred as Choice Points

(insert feature f2 of part_f into feature e2 of part_e).

(insert feature f1 of part_f into feature e2 of part_e).
```

Part_f is α-symmetric. So, application of the procedure for pruning the search tree using object symmetry finds that subgoals (sg1-1) and (sg1-3) may be equivalent subgoals provided features f1 and f2 are α-equivalent. Verifying that these features are α-equivalent, and finding that part_f is not used in any other assembly operation, the subgoal (sg1-3) is deleted from the search space altogether. Finding that subgoals (sg1-2) and (sg1-4) are in a similar relationship, the system deletes the subgoal (sg1-4). Thus, the subgoals of (G-1) after applying the object symmetry related procedures are:

```
Feature Level Subgoals

(insert feature f2 of part_f into feature e3 of part_e).
```

Feature Level Subgoals deferred as Choice Points

(insert feature f2 of part_f into feature e2 of part_e).

Although an initial confidence measure value of 0.5 is associated with both remaining subgoals (sg1-1) and (sg1-3), they are recognized to be equivalent. Thus, there is effectively one subgoal corresponding to goal (G-1). Therefore, the heuristics and subgoal ranking procedures are not applied to the goal. Thus, (sg1-1) is the hypothesized subgoal for (G-1). By an identical processing sequence, the hypothesized subgoal for (G-2) is

(sg2-1): insert feature g2 of part_g into feature e3 of part_e.

The hypothesized subgoals (sg1-1) and (sg2-1) are satisfactory refinements of goals (G-1) and (G-2), respectively, when the goals are considered in isolation. However, feature e3 is proposed as the receiving feature for both pegs, which is infeasible. In the accessibility analysis phase, the accessibility of e3 is a precondition of (G-2) and consequently is a postcondition of (G-1). The system infers that the accomplishment of (G-1) with the mating features specified in (sg1-1) violates this postcondition, while the accomplishment of the goal with the mating features specified in (sg1-3), which specified e2 in place of e3, does not violate the postcondition. Therefore, the alternative subgoal (sg1-3) is hypothesized as the interpretation of goal (G-1). This results in the following set of hypothesized subgoals at the completion of the accessibility analysis phase:

(insert feature f2 of part_f into feature e2 of part_e).

(insert feature g2 of part_g into feature e3 of part_e).

The following subfeatures corresponding to features f2, e2, g2, and e3 are generated based on the assembly operation and feature shape as outlined in Table 5:

feature	*subfeatures*
f2	f2_curved_face f2_bottom_face
g2	g2_curved_face g2_bottom_face
e2	e2_curved_face e2_bottom_face
e3	e3_curved_face e3_bottom_face

The following spatial relationships between these subfeatures are derived based on the assembly operation as per the procedure summarized in Table 6:

```
g2_botttom_face against e3_botttom_face
g2_curved_face fits e3_curved_face
f2_botttom_face against e2_botttom_face
f2_curved_face fits e2_curved_face.
```

The information pertaining to the subfeatures, as required by RAPT, including the spatial relationships inferred, are written into a file specified by the user. On processing this information, RAPT identifies the locations of the coordinate frames associated with part_f and part_g with respect to the object frame in part_e. The input file generated for RAPT, as well as the output produced by RAPT are provided in Sections A.3 and A.4 of Appendix A, respectively. The inferred locations of objects part_f and part_g in the resulting assembly, with respect to the coordinate frame in part_e are given by the following homogeneous transformations:

$$\text{location of part\_f} = \begin{bmatrix} 0 & 1 & 0 & 1.5 \\ 0 & 0 & 1 & 1.0 \\ 1 & 0 & 0 & 1.7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{location of part\_g} = \begin{bmatrix} 0 & 1 & 0 & -1.5 \\ 0 & 0 & 1 & 1.0 \\ 1 & 0 & 0 & 1.7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

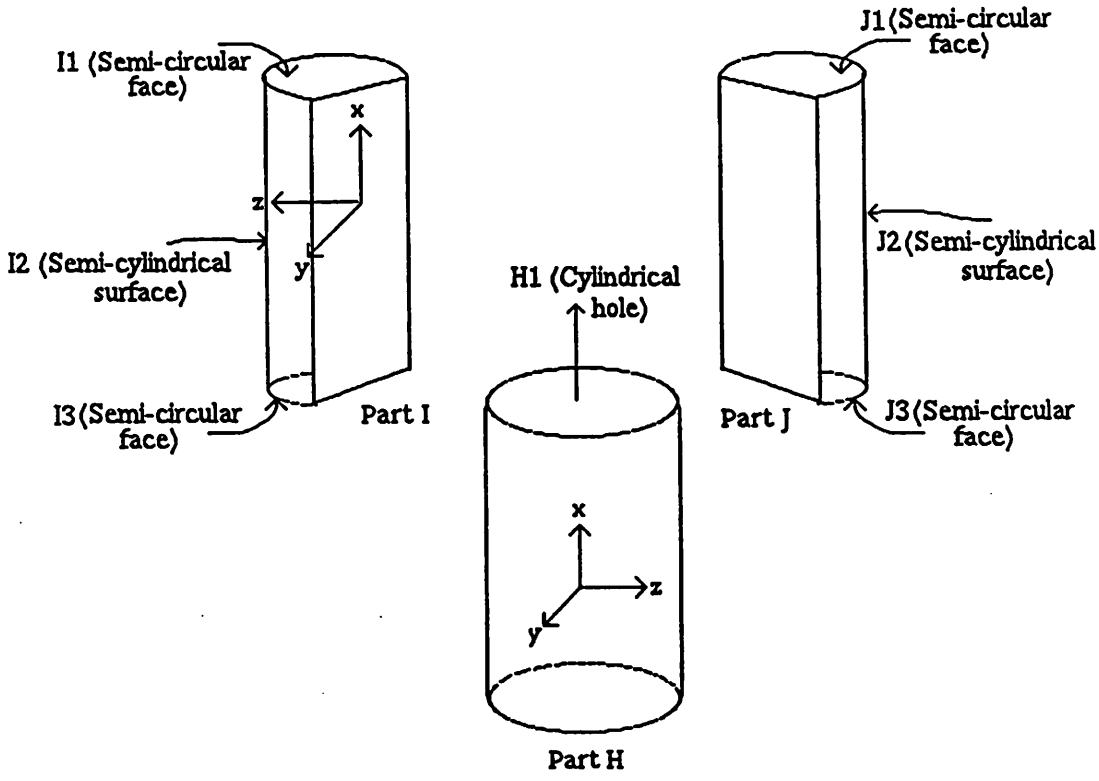


Figure 23: Two Volumes in the Same Feature

Although the rotational degree of freedom about the axes of the pegs is suppressed, it is easily verified from the specification of the object coordinate frames in Figure 21 and the position attributes of features e2, e3, f2 and g2 shown in Section A.1.1 that the transformation matrices above provide correct locations for the objects part_f and part_g.

7.2 Experiment 2: Two Semi-Cylinders in a Cylindrical Hole

The objects in this experimental assembly and their features are shown in Figure 23.

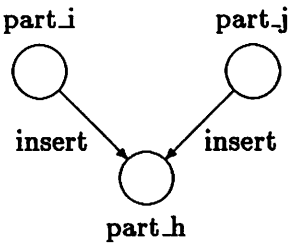


Figure 24: Assembly graph for the Two Volumes in the Same Feature Assembly

The graph indicating the complexity of the assembly is shown in Figure 24. The cylindrical container, part_h, is modeled as having a single feature: a cylindrical hole (h1). The semi-cylindrical peg, part_i, is modeled as having three features: two semi-circular faces i1 and i3, and a semi-cylindrical surface i2. Part_j has an identical set of features: two semi-circular faces j1 and j2, and a semi-cylindrical surface j3. The symmetry attributes of these objects are as shown below:

Object	α-symmetry	β-symmetry
part_h	2π	0
part_i	π	2π
part_j	π	2π

The object and feature information provided as input corresponding to the the three objects and their features listed above are shown in Section B.1.1. The coordinate frames embedded in objects, with respect to which feature positions are specified, are shown in Figure 23.

The assembly instructions provided as input to ASSEMBLE are:

(G-1): `Insert part_i into part_h`

(G-2): `Insert part_j into part_h`

(G-3): `(G-1) precedes (G-2).`

As in the previous experiment, the assembly operations and the characteristics of objects involved are identical for both goals. So, the processing of the FIND-FEATURES subsystem is similar for (G-1) and (G-2). The generation and pruning of the subtree rooted at (G-1) is presented here. A complete trace produced by ASSEMBLE for this experiment is provided in Section B.2.

Application of the geometric constraints associated with the INSERT operation results in the generation of the following feature-level subgoals corresponding to goal (G-1):

(sg1–1): `Insert feature i3 of part_i into feature h1 of part_h`

(sg1–2): `Insert feature i1 of part_i into feature h1 of part_h.`

Neither of the objects part_h and part_i exhibit finite β-symmetry. However, part_i is α-symmetric. Subgoals (sg1-1) and (sg1-2) are found to be equivalent by application of the procedure described in Section 6.5.2. The mating feature from part_h is the same on both subgoals, and the system verifies that features i1 and i3 of part_i are α-equivalent. As part_i is not being used in any other assembly operation, subgoal (sg1-2) is deleted from the search space leaving subgoal (sg1-1) as the only viable interpretation of goal (G-1). The subtree emanating from (G-1) has a single leaf node as shown below:

```
Feature Level Subgoals

(insert feature i3 of part_i into feature h1 of part_h).
```

As (sg1-1) is the only subgoal on the goal network corresponding to (G-1), a confidence measure value of 1.0 is associated with (sg1-1) and assembly heuristics are not applied. Subgoal (sg1-1), being the only choice, is the hypothesized subgoal proposed by the FIND-FEATURES subsystem. As a result of an identical processing sequence,

`(sg2-1): insert feature j3 of part_j into feature h1 of part_h`

is the hypothesized subgoal corresponding to goal (G-2).

In the accessibility analysis phase, the accessibility of h1 is a precondition for goal (G-2) and consequently, is a postcondition for goal (G-1). As described in page 92, determining whether the postcondition will be violated by execution of (G-1) given the

interpretation provided by (sg1-1) requires computing the effect of executing (G-1). On computing the area of cross-section of h1, the area of i3, and comparing them, the system infers that h1 is not consumed by the INSERT operation in (G-1). Therefore, the interaction analysis procedure concludes that the postcondition concerning the accessibility of h1 is not violated by (G-1). Although the inference is correct in the case of this experiment, clearly the reasoning is inadequate for it does not take into consideration the area of j3 nor the area of cross-section of the shape of h1 on completion of (G-1). The set of hypothesized subgoals at the completion of the accessibility analysis phase are:

```
(insert feature j3 of part_j into feature h1 of part_h).

(insert feature i3 of part_i into feature h1 of part_h).
```

The following subfeatures corresponding to features h1, i3 and j3 are generated based on the assembly operation and feature shape as outlined in Table 5:

feature	*subfeatures*
h1	h1_curved_face h1_bottom_face
i3	i3_curved_face i3_bottom_face
j3	j3_curved_face j3_bottom_face

Although features i3 and j3 are semi-circular faces, virtual cylindrical surfaces are assumed as the features are used in INSERT operations.

The following spatial relationships between these subfeatures are derived based on the procedure outlined in Table 6:

```
j3_botttom_face against h1_botttom_face
j3_curved_face  fits    h1_curved_face
i3_botttom_face against h1_botttom_face
i3_curved_face  fits    h1_curved_face.
```

The information required by RAPT, pertaining to these subfeatures, is generated. RAPT processes the information, finds the spatial relationships to be consistent, and

identifies the locations of the two semi-cylindrical pegs part_i and part_j with respect to part_h. The input to RAPT as well the output produced by RAPT for this experiment are shown in Sections B.3 and B.4 of Appendix B, respectively. The inferred locations of part_i and part_j with respect to the coordinate frame in part_h are given by the following homogeneous transformation matrices:

$$\text{location of part\_i} = \begin{bmatrix} 1 & 0 & 0 & 0.05 \\ 0 & 1 & 0 & 0.0 \\ 0 & 0 & 1 & -0.7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{location of part\_j} = \begin{bmatrix} 1 & 0 & 0 & 0.05 \\ 0 & 1 & 0 & 0.0 \\ 0 & 0 & 1 & -0.7 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The inferred locations for part_i and part_j are identical. The locations identified are satisfactory for the individual goals, as can be verified from the object coordinate frames shown in Figure 23 and the position attributes of features h1, i3 and i3 shown in Section B.1.1. However, the specified locations will result in objects part_i and part_j occupying coincident volumes in the cylindrical hole, which is infeasible.

7.3 Experiment 3: Double-Insertions Assembly Task

The objects involved in this experiment and their features are shown in Figure 25. One of the objects is a rectangular plate that has six circular holes, five of which are of the same dimension and one that is larger. each of the other three objects are to be inserted into the holes in the plate using DOUBLE-INSERT operations. The graph corresponding to the assembly indicating its complexity is shown in Figure 26.

The rectangular plate, part_k, is modeled as having seven features: a rectangular face (k1), a large cylindrical hole (k2), and five smaller cylindrical holes (k3-k7). The structures representing the object and its features are shown in Section C.1.1. The

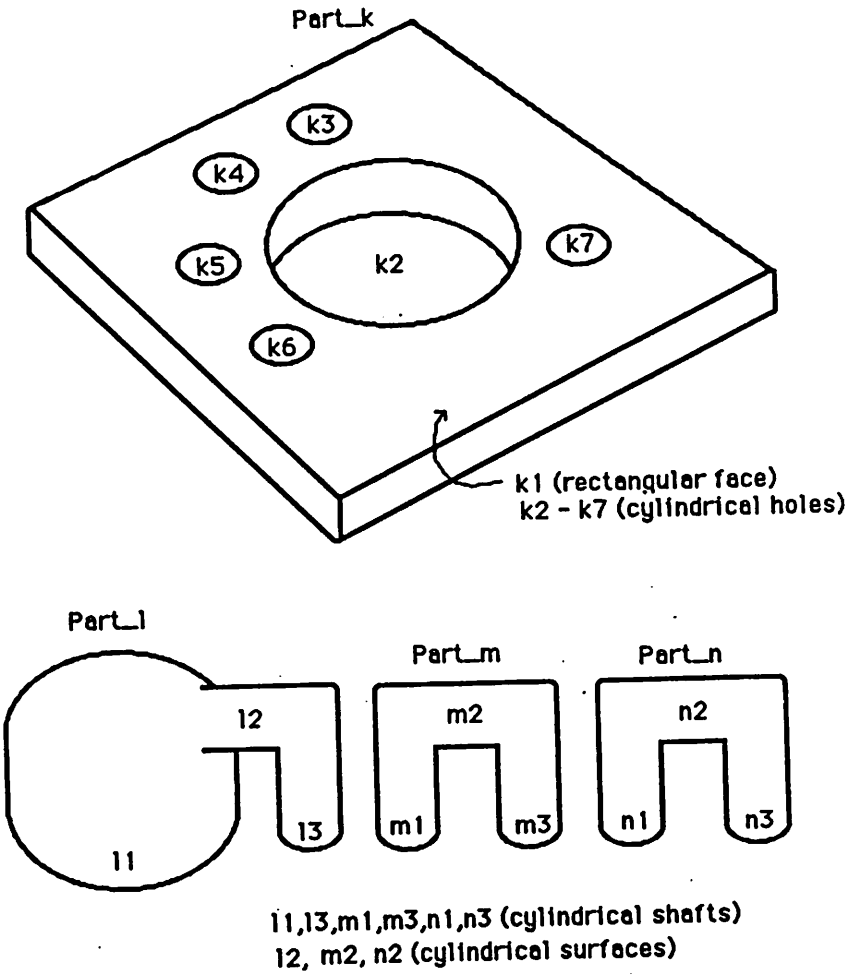


Figure 25: Double-Insertions

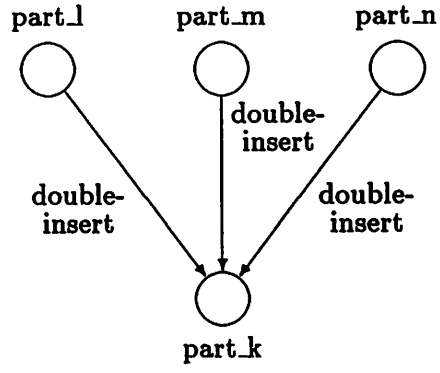


Figure 26: Assembly graph for the Double-insertions Assembly Task

positions of these features, as shown in these structures, indicate the distance relationships between the features. The feature k2 is equidistant from each of the features k3 through k7. The distances between features k3 and k4, k4 and k5, and k5 and k6 are equal. Each of the other three objects, part_l, part_m and part_n are modeled as having three features: two cylindrical shafts l1 and l3, m1 and m3, and n1 and n3, and a cylindrical surface that connects the two shafts, l2, m2 and n2. The symmetry attributes of these objects as represented in the object models are shown below:

Object	α-symmetry	β-symmetry
part_k	2π	2π
part_l	2π	2π
part_m	2π	π
part_n	2π	π

As seen from the table above, objects part_m and part_n are β-symmetric. The structures corresponding to objects part_l, part_m and part_n are shown in Section C.1.1. The distance between l1 and l3 is the same as the distance between k2 and each of the features k3 through k7. The distance between m1 and m3 (and the distance between n1 and n3) is the same as the distance between features k3 and k4 (and k4 and k5, and k5 and k6).

The assembly instructions given as input to ASSEMBLE are:

(G–1): Double-insert part_l in part_k

(G–2): Double-insert part_m in part_k

(G–3): Double-insert part_n in part_k

(G–4): (G-1) precedes (G-2) and (G-2) precedes (G-3).

The processing of goals (G-1) and (G-2) by the FIND-FEATURES subsystem is presented here. As the assembly operation and the objects involved in (G-2) and (G-3) are identical, detailed account of the processing of (G-3) is omitted. The results of the subsystem for goal (G-3) are presented. A complete trace of the processing of ASSEMBLE on this experiment is provided in Section C.2.

Application of the geometric constraints associated with the DOUBLE-INSERT operation, including the distance constraint and relative orientation constraint, results in the generation of the following feature-level subgoals corresponding to goal (G-1):

(sg1–1): Double-insert feature l1 of part_l in feature k2 of part_k
and feature l3 in feature k7

(sg1–2): Double-insert feature l1 of part_l in feature k2 of part_k
and feature l3 in feature k6

(sg1–3): Double-insert feature l1 of part_l in feature k2 of part_k
and feature l3 in feature k5

(sg1–4): Double-insert feature l1 of part_l in feature k2 of part_k
and feature l3 in feature k4

(sg1–5): Double-insert feature l1 of part_l in feature k2 of part_k
and feature l3 in feature k3.

The mating feature for the larger shaft, l1, is uniquely determined as the larger cylindrical hole, k2. The smaller shaft, l3, can go into any of the five smaller holes on part_k, k3 through k7, while still respecting the distance and relative orientation constraints.

Neither of the objects part_k and part_l have finite symmetry. So the set of subgoals of (G-1) are unaffected by the procedures that utilize object symmetry to prune the search tree. An initial confidence measure value of 0.2 is assigned to each of the subgoals (sg1-1)

through (sg1-5). As each of the cylindrical holes k3 through k7 are identical in size, the closeness-of-fit heuristic does not alter the confidence measure values of the subgoals. The functional symmetry heuristic recognizes that features k3, k4, k5, k6 and k7 are functionally symmetric and therefore that subgoals (sg1-1) through (sg1-5) fall into the same equivalence class. All subgoals except (sg1-1) are moved to the list of choice points. The FIND-FEATURES subsystem produces (sg1-1) as the hypothesized subgoal for goal (G-1).

In the case of goal (G-2), application of the operator related constraints results in the generation of the following 16 subgoals:

(sg2–1): Double-insert feature m3 of part_m in feature k5 of part_k
and feature m1 in feature k6

(sg2–2): Double-insert feature m1 of part_m in feature k5 of part_k
and feature m3 in feature k6

(sg2–3): Double-insert feature m3 of part_m in feature k4 of part_k
and feature m1 in feature k5

(sg2–4): Double-insert feature m1 of part_m in feature k4 of part_k
and feature m3 in feature k5

(sg2–5): Double-insert feature m3 of part_m in feature k3 of part_k
and feature m1 in feature k4

(sg2–6): Double-insert feature m1 of part_m in feature k3 of part_k
and feature m3 in feature k4

(sg2–7): Double-insert feature m3 of part_m in feature k2 of part_k
and feature m1 in feature k7

(sg2–8): Double-insert feature m1 of part_m in feature k2 of part_k
and feature m3 in feature k7

(sg2–9): Double-insert feature m3 of part_m in feature k2 of part_k
and feature m1 in feature k6

(sg2–10): Double-insert feature m1 of part_m in feature k2 of part_k
and feature m3 in feature k6

(sg2–11): Double-insert feature m3 of part_m in feature k2 of part_k
and feature m1 in feature k5

(sg2–12): Double-insert feature m1 of part_m in feature k2 of part_k
and feature m3 in feature k5

(sg2–13): Double-insert feature m3 of part_m in feature k2 of part_k and feature m1 in feature k4

(sg2–14): Double-insert feature m1 of part_m in feature k2 of part_k and feature m3 in feature k4

(sg2–15): Double-insert feature m3 of part_m in feature k2 of part_k and feature m1 in feature k3

(sg2–16): Double-insert feature m1 of part_m in feature k2 of part_k and feature m3 in feature k3.

The features m1 and m3 can be inserted into k3 and k4, k4 and k5, and k5 and k6 or one of the features can be inserted into the larger hole k2 and the other into one of the five smaller holes k3 through k7. All of these possibilities, listed above, satisfy the distance and relative orientation constraints.

Object part_m is β-symmetric and consequently features m1 and m3 are β-equivalent. In each of the eight pairs of subgoals listed above, the subgoals of the pair only differ in that the roles of features m1 and m3 are switched around. Application of the procedures that prune the set of subgoals based on object symmetry reduces the set of subgoals of (G-2) to the following:

```
(sg2-1): (double-insert feature m3 of part_m in feature k5
                  and feature m1 in feature k6).

(sg2-3): (double-insert feature m3 of part_m in feature k4
                  and feature m1 in feature k5).

(sg2-5): (double-insert feature m3 of part_m in feature k3
                  and feature m1 in feature k4).

(sg2-7): (double-insert feature m3 of part_m in feature k2
                  and feature m1 in feature k7).

(sg2-9): (double-insert feature m3 of part_m in feature k2
                  and feature m1 in feature k6).

(sg2-11): (double-insert feature m3 of part_m in feature k2
                  and feature m1 in feature k5).

(sg2-13): (double-insert feature m3 of part_m in feature k2
```

```
                and feature m1 in feature k4).

(sg2-15): (double-insert feature m3 of part_m in feature k2
                and feature m1 in feature k3).
```

An initial confidence measure value of 0.125 is assigned to each of the eight remaining subgoals. Of the eight, five of the subgoals involve the larger cylindrical hole k2 of part_k. The remaining three that use a pair of smaller holes of part_k represent a closer fit than the five that use k2. The closeness-of-fit heuristic partitions the set of subgoals into two sets P and D where

$$P = \{(\text{sg2-1}),(\text{sg2-3}),(\text{sg2-5})\}$$

and

$$D = \{(\text{sg2-7}),(\text{sg2-9}),(\text{sg2-11}),(\text{sg2-13}),(\text{sg2-15})\}.$$

The confidence measure values of the subgoals in P are increased and those of subgoals in D are correspondingly decreased as per the procedure described in page 86. The eight subgoals with the resulting confidence measure values are:

Feature Level Subgoals	Confidence Rates
(double-insert feature m3 of part_m in feature k5 of part_k and feature m1 in feature k6)	0.22916666
(double-insert feature m3 of part_m in feature k4 of part_k and feature m1 in feature k5)	0.22916666
(double-insert feature m3 of part_m in feature k3 of part_k and feature m1 in feature k4)	0.22916666
(double-insert feature m3 of part_m in feature k2 of part_k and feature m1 in feature k7)	0.0625
(double-insert feature m3 of part_m in feature k2 of part_k and feature m1 in feature k6)	0.0625
(double-insert feature m3 of part_m in feature k2 of part_k and feature m1 in feature k5)	0.0625
(double-insert feature m3 of part_m in feature k2 of part_k	

```
          and feature m1 in feature k4)                    0.0625

(double-insert feature m3 of part_m in feature k2 of part_k
          and feature m1 in feature k3)                    0.0625
```

As the cylindrical holes k3 through k7 are of identical size, the functional symmetry heuristic recognizes two equivalence classes: one that comprises subgoals (sg2-1), (sg2-3) and (sg2-5), and another which comprises subgoals (sg2-7), (sg2-9), (sg2-11), (sg2-13) and (sg2-15). As representative elements from each equivalence class, subgoals (sg2-1) and (sg2-7) are retained in the list of subgoals and the remaining eight subgoals are moved to the list of choice points. The set of subgoals of (G-2) remaining on its subgoals list are:

```
Feature Level Subgoals                                     Confidence
                                                             Rates

(double-insert feature m3 of part_m in feature k5 of part_k
          and feature m1 in feature k6)                    0.22916666

(double-insert feature m3 of part_m in feature k2 of part_k
          and feature m1 in feature k7)                    0.0625
```

The hypothesis selection process selects subgoal (sg2-1) as the hypothesized subgoal for goal (G-2) as it has the higher confidence measure value. By a similar processing sequence:

```
(sg3-1): Double-insert feature n3 of part_n in feature k5 of part_k
                  and feature n1 in feature k6
```

is chosen as the hypothesized subgoal for goal (G-3).

Features k5 and k6 are specified in the hypothesized subgoals of (G-2) and (G-3). The accessibility of features k5 and k6 are among the preconditions of (G-3) and consequently are postconditions of (G-2). The accessibility analysis phase recognizes that the choice of mating features for (G-2) would violate this postcondition, searches through the list of choice points for a subgoal that would not violate the postconditions of (G-2) and selects subgoal (sg2-5) as an alternate hypothesis. Thus, the set of hypothesized subgoals for the goals at the completion of accessibility analysis are:

```
(double-insert feature n3 of part_n in feature k5 of part_k
          and feature n1 in feature k6).

(double-insert feature m3 of part_m in feature k3 of part_k
          and feature m1 in feature k4).

(double-insert feature l1 of part_l in feature k2 of part_k
          and feature l3 in feature k7).
```

The subfeatures corresponding to features k2 through k7 of part_k, and features m1 and m3, and n1 and n3 of part_m and part_n, respectively, are generated based on Table 5. The following spatial relationships between these subfeatures are derived based on Table 6:

```
n3_botttom_face against k5_botttom_face
n3_curved_face  fits    k5_curved_face
n1_botttom_face against k6_botttom_face
n1_curved_face  fits    k6_curved_face
m3_botttom_face against k3_botttom_face
m3_curved_face  fits    k3_curved_face
m1_botttom_face against k4_botttom_face
m1_curved_face  fits    k4_curved_face
l1_botttom_face against k2_botttom_face
l1_curved_face  fits    k2_curved_face
l3_botttom_face against k7_botttom_face
l3_curved_face  fits    k7_curved_face.
```

The information required by RAPT, pertaining to these subfeatures, is generated. RAPT processes the information, finds the spatial relationships to be consistent, and identifies the locations of objects part_l, part_m, and part_n with respect to the coordinate frame in part_k. The input to RAPT as well the output produced by RAPT for this experiment are shown in SectionsC.3 and C.4 of Appendix C. The inferred locations of part_l, part_m and part_n with respect to the coordinate frame in part_k are given by the following homogeneous transformation matrices:

$$\text{location of part\_l} = \begin{bmatrix} 0.7 & 0.7 & 0 & 1.189 \\ -0.7 & 0.7 & 0 & -1.189 \\ 0 & 0 & 1 & 2.25 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{location of part\_m} = \begin{bmatrix} 0 & 0 & 1 & -2.41 \\ 0 & -1 & 0 & 0.0 \\ 1 & 0 & 0 & 2.25 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{location of part\_n} = \begin{bmatrix} 0 & -1 & 0 & 0.0 \\ 0 & 0 & -1 & 2.41 \\ 1 & 0 & 0 & 2.25 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It has been verified from the position attributes of features and the specification of the object coordinate frames that the inferred locations are accurate.

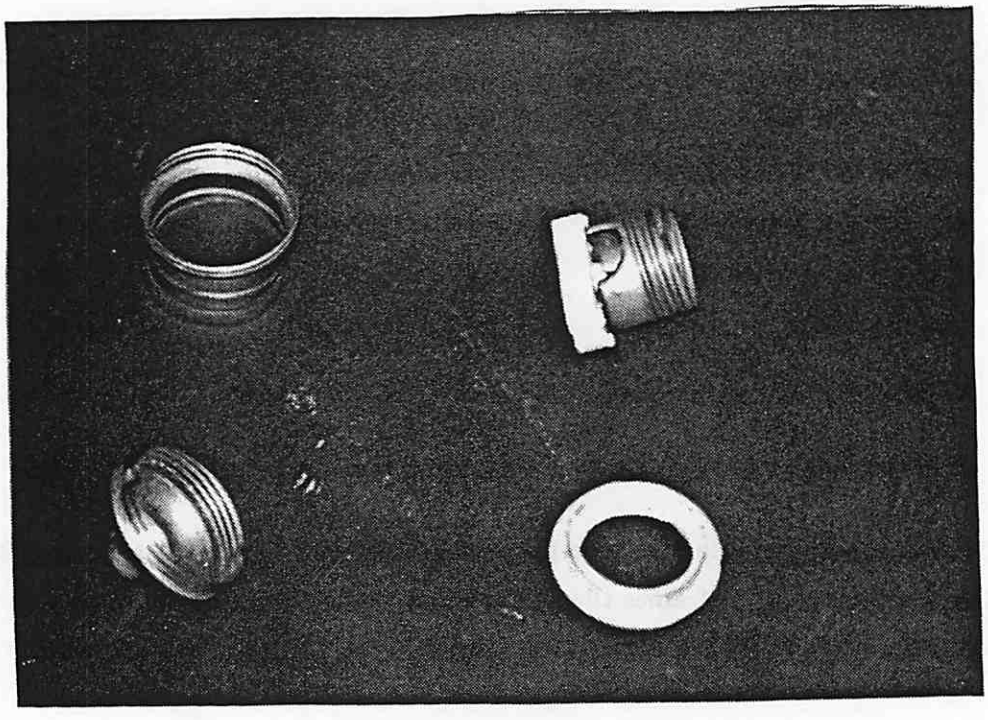


Figure 27: Components of the Light Bulb Fixture Assembly

7.4 Experiment 4: The Lightbulb Fixture Assembly Task

The objects involved in the assembly and the features of these objects that were modeled are shown in Figure 5. The graph corresponding to the assembly indicating its complexity is shown in Figure 28. The object, part_a, is hollow, and cylindrical on both ends with one end much narrower than the other. It is threaded internally on the narrow end and externally on the wider end. The object is modeled using four features: a1, a cylindrical hole at the narrow end; a2, the threading at the narrow end; a3, the wider cylindrical hole; and a4, the threading at the wide end. Part_b is composed of two shapes: a solid cuboid and a hollow cylinder. The features of part_b that are modeled are: b1, the rectangular plane surface of the cuboid farthest from the cylinder; b2, the cylindrical hole; b3, the external threading around the cylindrical hole b2; and b4, the outer cylindrical surface of the hollow cylinder. Part_c has the appearance of a hollow cylinder, open at both ends. It is wider at one end than the other. Its wider end is threaded internally. This object is modeled as having three features: c1, the wider cylindrical hole; c2, the threading around the wider end; and c3, the narrower cylindrical hole. The object, part_d,

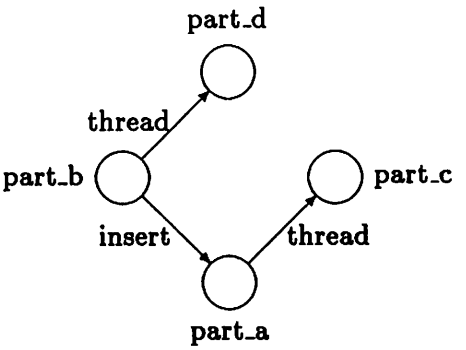


Figure 28: Assembly graph for the Light Bulb Fixture Assembly

is a hollow cylinder with a small height. It is threaded internally all the way. The object has an obtruding lip at one end. The object is modeled as having four features: two cylindrical holes, d1 and d3, and two threaded features d2 and d4. While, d1 and d3, as well as d2 and d4 are overlapping features, they are modeled distinctly to reflect the fact that the object can be oriented in either direction.

The symmetry attributes of these objects as represented in the object models are shown below:

Object	α-symmetry	β-symmetry
part_a	2π	π
part_b	2π	π
part_c	2π	0
part_d	2π	0

The structures corresponding to the objects, and their features, that constitute part of the input information to ASSEMBLE, are shown in Section D.1.1. As is seen from the symmetry attributes of these objects, part_a and part_b are modeled as having finite β-symmetry, while part_c and part_d have infinite symmetry about their principal axes.

The assembly instructions given as input to ASSEMBLE are:

(G–1): `Insert part_b in part_a`

(G–2): `Thread part_a with part_c`

(G-3): Thread part_b with part_d

(G-4): (G-1) precedes (G-2) and (G-2) precedes (G-3).

The FIND-FEATURES subsystem processes each of these goals in succession. A brief description of the reasoning process by which the hypothesized subgoals for each of the goals listed above is determined is presented here. A complete trace produced by ASSEMBLE for this experiment is provided in Section D.2.

Application of the constraints associated with the INSERT operation results in the generation of the following feature-level subgoals corresponding to goal (G-1):

(sg1-1): Insert feature b4 of part_b in feature a3 of part_a

(sg1-2): Insert feature b2 of part_b in feature a3 of part_a

(sg1-3): Insert feature b1 of part_b in feature a3 of part_a.

The larger cylindrical hole, a3, of part_a is uniquely identified as the mating feature of that object. Part_b, however, can be inserted in several different orientations. Although objects part_a and part_b have finite β-symmetry, the mating features considered are not β-equivalent. Therefore, application of the procedures relating to object symmetry do not effect any changes in the set of subgoals.

An initial confidence measure value of 0.3333 is associated with each of the subgoals (sg1-1), (sg1-2) and (sg1-3). Subgoal (sg1-3) represents a closer fit than the other two alternatives. Therefore, application of the closeness-of-fit heuristic alters the confidence measure values of the subgoals as follows:

Feature Level Subgoals	Confidence Rates
(insert feature b4 of part_b in feature a3 of part_a)	0.16666667
(insert feature b2 of part_b in feature a3 of part_a)	0.16666667
(insert feature b1 of part_b in feature a3 of part_a)	0.6666667

The distinguishing-feature heuristic is applicable to this goal as there is a single subgoal (sg1-3) that has the highest confidence measure value. The purpose underlying

the heuristic is the acquisition of sensory information that can reinforce the hypothesis represented by the highest ranking subgoal. In the ASSEMBLE system, such sensory capability is simulated by hand-coding information specific to this experiment that augments the object and feature information associated with part_a, part_b, a3 and b1. The flanges and grooves on part_a and part_b are added to list of features associated with these objects and the feature graphs of these objects are appropriately modified. The augmented structures corresponding to part_a, a3, part_b and b1 are shown below:

```
#<object 7243377> is a structure of type object
   NAME:                          part_a
   SHAPE:                         cylinder
   LENGTH:                        3.6
   WIDTH:                         3.6
   HEIGHT:                        2.4
   ALPHA:                         #(0 1 0 6.28318530717958 7d0)
   BETA:                          #(0 0 1 3.14159265358979 36d0)
   FEATURES:                      (a1 a2 a3 a4 a5 a6)

#<feature 7243737> is a structure of type feature
   ADJACENT-FEATURES:             (a4)
   INCLUSIVE-FEATURES:            (a5 a6)
   ENCOMPASSING-FEATURE:          nil
   OVERLAPPING-FEATURES:          nil
   COAXIAL-FEATURES:              (a4)
   FEATURE-NAME:                  a3
   FEATURE-FUNCTION:              container
   FEATURE-SHAPE:                 cylindrical-hole
   FEATURE-DIMENSIONS:            (3.6 2.0)
   FEATURE-POSITION:              #2A((0 0 1.2 1.0) (0 0 1 0))
   FEATURE-BETA:                  3.14159265358979 36d0
   SUB-FEATURES:                  nil

#<object 7244256> is a structure of type object
   NAME:                          part_b
   SHAPE:                         box-plus-cylinder
   LENGTH:                        3.2
   WIDTH:                         3.1
   HEIGHT:                        1.6
   ALPHA:                         #(1 0 0 6.28318530717958 7d0)
   BETA:                          #(0 1 0 3.14159265358979 36d0)
   FEATURES:                      (b1 b2 b3 b4 b6 b7)

#<feature 7244373> is a structure of type feature
```

```
ADJACENT-FEATURES:          nil
INCLUSIVE-FEATURES:         (b6 b7)
ENCOMPASSING-FEATURE:       nil
OVERLAPPING-FEATURES:       nil
COAXIAL-FEATURES:           nil
FEATURE-NAME:               b1
FEATURE-FUNCTION:           insertor
FEATURE-SHAPE:              rectangular-face
FEATURE-DIMENSIONS:         (3.2 1.6)
FEATURE-POSITION:           #2A((0 0 -1.6 1.0) (0 0 -1 0))
FEATURE-BETA:               3.141592653897936d0
SUB-FEATURES:               nil
```

The structures corresponding to the newly included features are shown in Section D.1.2. As the shape and dimensions of these newly included features correspond, the heuristic results in modifying the confidence measure values of all the subgoals as follows:

`(insert feature b4 of part_b in feature a3 of part_a)` 0.083333336

`(insert feature b2 of part_b in feature a3 of part_a)` 0.083333336

`(insert feature b1 of part_b in feature a3 of part_a)` 0.8333334

The functional symmetry heuristic does not apply as the different features of part_b have differing characteristics. The subgoal selection process chooses (sg1-3) as the hypothesized subgoal as it has the highest confidence measure value and moves the other two subgoals to the list of choice-points.

Application of the constraints associated with the THREAD operation results in the identification of the single subgoal:

(sg2–1): `Thread feature a4 of part_a in feature c2 of part_c.`

Comparison of the diameters and pitch of threads allows unique identification of the mating features in this case. No further processing is required to generate the hypothesized subgoal. A confidence measure value of 1.0 is associated with subgoal (sg2-1) which serves as the hypothesized subgoal for goal (G-2).

In the case of goal (G-3), application of operator related constraints generate the following two subgoals:

(sg3–1): Thread feature b3 of part_b with feature d4 of part_d

(sg3–2): Thread feature b3 of part_b with feature d2 of part_d.

Feature b3 from part_b is uniquely identified as the mating feature, while features d2 and d4 from part_d are identified as possible mating features corresponding to the two different orientations in which the threading can occur. Procedures relating to object symmetry have no effect on the set of subgoals. An initial confidence measure value of 0.5 is assigned to each of the subgoals (sg3-1) and (sg3-2). The functional symmetry heuristic considers the subgoals to be equivalent and moves subgoal (sg3-2) to the list of choice points, leaving subgoal (sg3-1) as the hypothesized subgoal for goal (G-3).

The set of hypothesized subgoals determined by the FIND-FEATURES subsystem are:

```
(thread feature b3 of part_b with feature d4 of part_d).

(thread feature a4 of part_a with feature c2 of part_c).

(insert feature b1 of part_b in feature a3 of part_a).
```

The accessibility analysis phase includes the accessibility of a4 as a postcondition for goal (G-1). The interaction analysis procedure concludes that the postcondition is not violated by the choice of mating features for goal (G-1) and no change in the goal network results.

The following subfeatures corresponding to features a3, a4, b1, c2, and d4 are generated based on the assembly operation and feature shape as outlined in Table 5:

feature	*subfeatures*
a3	a3_curved_face a3_bottom_face
a4	a4_curved_face a4_bottom_face
b1	b1_curved_face b1_bottom_face
b3	b3_curved_face b3_bottom_face
c2	c2_curved_face c2_bottom_face
d4	d4_curved_face d4_bottom_face

The following spatial relationships between these subfeatures are derived based on the procedure outlined in Table 6:

```
b3_botttom_face against d4_botttom_face
b3_curved_face  fits    d4_curved_face
a4_botttom_face against c2_botttom_face
a4_curved_face  fits    c2_curved_face
b1_botttom_face against a3_botttom_face
b1_curved_face  fits    a3_curved_face.
```

The information required by RAPT, pertaining to these subfeatures, is generated. RAPT processes the information, finds the spatial relationships to be consistent, and identifies the locations of objects part_b, part_c and part_d with respect to the coordinate frame associated with part_a. The input to RAPT as well the output produced by RAPT for this experiment are shown in Sections D.3 and D.4 of Appendix D, respectively. The inferred locations of part_b, part_c and part_d are given by the following homogeneous transformation matrices:

$$\text{location of part\_b} = \begin{bmatrix} 0 & 0 & 1 & 0.0 \\ 1 & 0 & 0 & 0.0 \\ 0 & 1 & 0 & 2.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{location of part\_c} = \begin{bmatrix} 1 & 0 & 0 & 0.0 \\ 0 & 1 & 0 & 0.0 \\ 0 & 0 & 1 & 1.8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{location of part\_d} = \begin{bmatrix} 1 & 0 & 0 & 0.0 \\ 0 & 1 & 0 & 0.0 \\ 0 & 0 & 1 & 3.0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The inferred locations for part_b and part_c are feasible. However, the hypothesized subgoal for goal (G-3) is incorrect. It results in a situation where the threading of part_d with part_b cannot be completed because part_c is in the way. Consequently, the inferred location of part_d is one that cannot be realized. Like experiment 2, this experiment also illustrates the need for analysis of object interference.

7.5 Experiment 5: The NC-100 Controller Assembly

As shown in Figure 29, the controller assembly is comprised of the following objects: a base_plate, a transformer, a capacitor, four small bolts, two side_plates, a top_plate and four large bolts. The complexity of the assembly is depicted in two subgraphs shown in Figure 30. The node representing the base_plate joins the two subgraphs to produce the complexity graph of the entire assembly. Unlike the complexity graphs of earlier assembly experiments, all of which have a complexity measure of 0 (i.e., they have no cycles), the graph corresponding to the controller assembly (to be specific, the undirected graph that is derived from joining the directed subgraphs shown in Figure 30) has several cycles of length 3 or more.

Recall that RAPT works with a graph representation of assembly tasks where the objects form the nodes and the spatial relationships form the arcs (see Section 5.3). The inferencing mechanism in RAPT works by detecting cycles of length 2 in this spatial

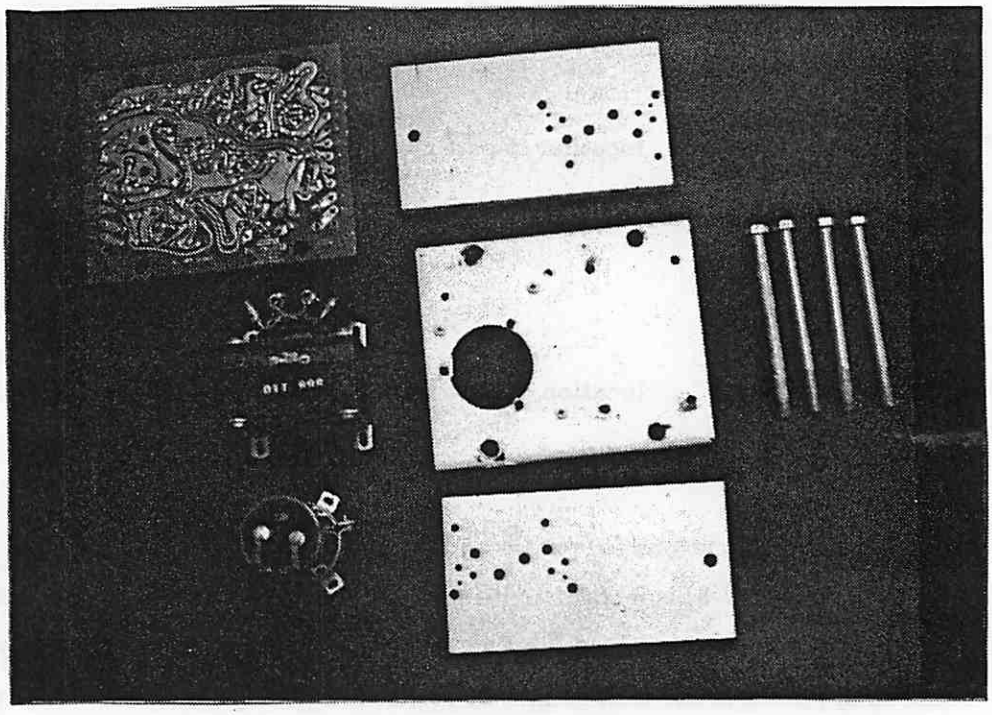


Figure 29: Components of the NC-100 Controller Assembly

relations graph and reducing the cycle to a single arc, or collapsing the nodes to a single node. Secondly, the system introduces additional arcs to cycles of length 3, where the additional arc in turn generates a cycle of length 2. This experiment exercises the component of RAPT that detects and utilizes cycles of length 3 on the spatial relations graph, while the other experiments produce spatial relations graphs containing only cycles of length 2.

The base_plate is modeled as having 10 features: the top surface on which the transformer and the side_plates will be placed; the large hole into which the capacitor will be inserted; the four larger holes symmetrically placed on the top surface into which the large bolts will be inserted and threaded after aligning the side_plates and top_plate; and the four small holes used for affixing the transformer to the base_plate. The transformer is modeled as having six features: its top and bottom surfaces and the four holes on its sides that will be aligned with the small holes in the base_plate. The capacitor is modeled as having two features: a cylindrical shaft and its top, a circular surface. Each of the side_plates are modeled as having seven features: three plane surfaces, and four

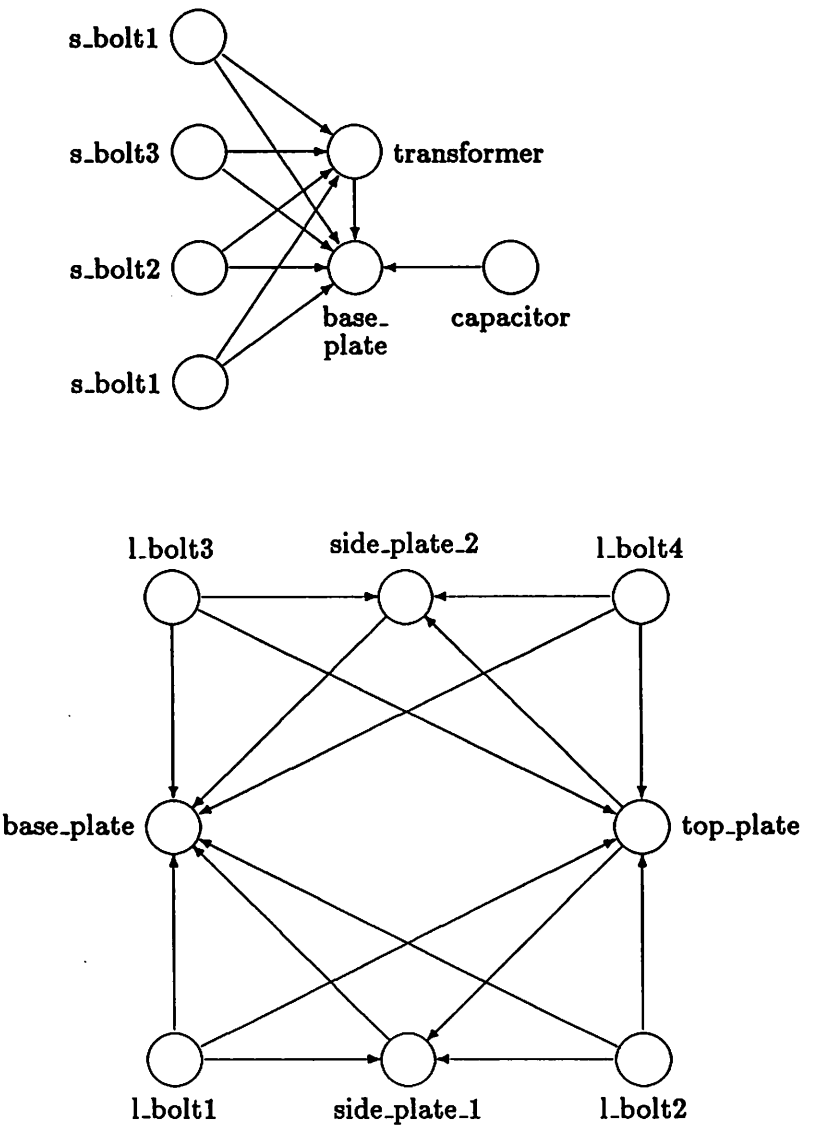


Figure 30: Assembly graph for the NC-100 controller assembly

holes which are aligned with the holes in top_plate and base_plate. The small bolts are represented as comprising of a shaft and a thread. The top_plate is modeled as having five features: its bottom surface which comes into contact with the side_plates; and four holes which are aligned with the holes in the side_plates and base_plate. Each of the large bolts are modeled as having three features: a head, a shaft and a thread. The structures corresponding to the objects and their features that constitute part of the input information to ASSEMBLE are shown in Section E.1.1.

The symmetry attributes of these objects as represented in the object models are shown below:

Object	α-symmetry	β-symmetry
base_plate	2π	2π
transformer	2π	π
capacitor	2π	π
side_plates	π	2π
top_plate	2π	2π
large bolts	2π	$\pi/3$
small bolts	2π	0

The assembly instructions given as input to ASSEMBLE and the precedence relations specified between these assembly operations are shown in Section E.1 of Appendix E. The input assembly instructions are grouped such that goals that involve the same or identical objects, such as:

```
thread l_bolt4 into base_plate
thread l_bolt3 into base_plate
```

are grouped together. This experiment brings out the usefulness of the spatial reasoning component of ASSEMBLE. Due to the large size and complexity of this experimental assembly task, the discussion in the present section focuses mainly on the behavior of the spatial reasoning component on this assembly task. The processing of this task by the FIND-FEATURES subsystem is not presented here. The trace outlined in Section E.2 of Appendix E provides sufficient information to the reader to gather the reasoning process by which the hypothesized subgoals are generated. The hypothesized subgoals generated

by the FIND-FEATURES subsystem for the goals associated with the controller assembly task are listed below:

```
(thread feature lb4_thread of l_bolt4 into feature lh4
                                          of base_plate)
(thread feature lb3_thread of l_bolt3 into feature lh4
                                          of base_plate)
(thread feature lb2_thread of l_bolt2 into feature lh4
                                          of base_plate)
(thread feature lb1_thread of l_bolt1 into feature lh4
                                          of base_plate)

(insert feature lb4_shaft of l_bolt4 into feature sp2_h4
                                          of side_plate_2)
(insert feature lb3_shaft of l_bolt3 into feature sp2_h4
                                          of side_plate_2)

(insert feature lb2_shaft of l_bolt2 into feature sp1_h4
                                          of side_plate_1)
(insert feature lb1_shaft of l_bolt1 into feature sp1_h4
                                          of side_plate_1)

(insert feature lb4_shaft of l_bolt4 into feature tp_h4
                                            of top_plate)
(insert feature lb3_shaft of l_bolt3 into feature tp_h4
                                            of top_plate)
(insert feature lb2_shaft of l_bolt2 into feature tp_h4
                                            of top_plate)
(insert feature lb1_shaft of l_bolt1 into feature tp_h4
                                            of top_plate)

(place feature tp_bottom of top_plate on feature sp2_face3
                                          of side_plate_2)
(place feature tp_bottom of top_plate on feature sp1_face3
                                          of side_plate_1)

(place feature sp2_face3 of side_plate_2 on feature plate_top
                                               of base_plate)
(place feature sp1_face3 of side_plate_1 on feature plate_top
                                               of base_plate)

(insert feature c_side of capacitor into feature hole
                                        of base_plate)

(thread feature sb4_thread of s_bolt4 into feature sh4
```

```
                                          of base_plate)
(thread feature sb3_thread of s_bolt3 into feature sh4
                                          of base_plate)
(thread feature sb2_thread of s_bolt2 into feature sh4
                                          of base_plate)
(thread feature sb1_thread of s_bolt1 into feature sh4
                                          of base_plate)

(insert feature sb4_shaft of s_bolt4 into feature h4
                                         of transformer)
(insert feature sb3_shaft of s_bolt3 into feature h4
                                         of transformer)
(insert feature sb2_shaft of s_bolt2 into feature h4
                                         of transformer)
(insert feature sb1_shaft of s_bolt1 into feature h4
                                         of transformer)

(place feature bottom of transformer on feature plate_top
                                              of base_plate)
```

Let $G_1 = (op \quad x \quad z)$ and $G_2 = (op \quad y \quad z)$ denote two distinct goals where *op* denotes a specific assembly operation,and x, y, and z denote objects. It is evident from the hypothesized subgoals listed above that, if x and y are identical, yet distinct objects, then the FIND-FEATURES subsystem proposes the same feature of z as a mating feature in the hypothesized subgoals of G_1 and G_2. The accessibility analysis phase corrects this situation and alters the set of hypothesized subgoals as follows:

```
(thread feature lb4_thread of l_bolt4 into feature lh4
                                          of base_plate)
(thread feature lb3_thread of l_bolt3 into feature lh3
                                          of base_plate)
(thread feature lb2_thread of l_bolt2 into feature lh2
                                          of base_plate)
(thread feature lb1_thread of l_bolt1 into feature lh1
                                          of base_plate)

(insert feature lb4_shaft of l_bolt4 into feature sp2_h4
                                          of side_plate_2)
(insert feature lb3_shaft of l_bolt3 into feature sp2_h1
                                          of side_plate_2)

(insert feature lb2_shaft of l_bolt2 into feature sp1_h4
```

```
          and feature n1 into feature k4)                0.125

(double-insert feature n3 of part_n into feature k2 of part_k
          and feature n1 into feature k7)                0.125

(double-insert feature n3 of part_n into feature k2 of part_k
          and feature n1 into feature k6)                0.125

(double-insert feature n3 of part_n into feature k2 of part_k
          and feature n1 into feature k5)                0.125

(double-insert feature n3 of part_n into feature k2 of part_k
              and feature n1 into feature k4)            0.125

(double-insert feature n3 of part_n into feature k2 of part_k
              and feature n1 into feature k3)            0.125


Heuristic closeness-of-fit modifies the confidence
rates of subgoals as follows:

(double-insert feature n3 of part_n into feature k5 of part_k
          and feature n1 into feature k6)            0.22916666

(double-insert feature n3 of part_n into feature k4 of part_k
          and feature n1 into feature k5)            0.22916666

(double-insert feature n3 of part_n into feature k3 of part_k
          and feature n1 into feature k4)            0.22916666

(double-insert feature n3 of part_n into feature k2 of part_k
          and feature n1 into feature k7)                0.0625

(double-insert feature n3 of part_n into feature k2 of part_k
          and feature n1 into feature k6)                0.0625

(double-insert feature n3 of part_n into feature k2 of part_k
          and feature n1 into feature k5)                0.0625

(double-insert feature n3 of part_n into feature k2 of part_k
          and feature n1 into feature k4)                0.0625

(double-insert feature n3 of part_n into feature k2 of part_k
          and feature n1 into feature k3)                0.0625
```

Functional Symmetry:

Subgoal
(double-insert feature n3 of part_n into feature k4 of part_k
and feature n1 into feature k5)
is moved to the list of choice points. There is a functionally symmetrical alternative in
(double-insert feature n3 of part_n into feature k5 of part_k
and feature n1 into feature k6).

Subgoal
(double-insert feature n3 of part_n into feature k3 of part_k
and feature n1 into feature k4)
is moved to the list of choice points. There is a functionally symmetrical alternative in
(double-insert feature n3 of part_n into feature k5 of part_k
and feature n1 into feature k6).

Subgoal
(double-insert feature n3 of part_n into feature k2 of part_k
and feature n1 into feature k6)
is moved to the list of choice points. There is a functionally symmetrical alternative in
(double-insert feature n3 of part_n into feature k2 of part_k
and feature n1 into feature k7).

Subgoal
(double-insert feature n3 of part_n into feature k2 of part_k
and feature n1 into feature k5)
is moved to the list of choice points. There is a functionally symmetrical alternative in
(double-insert feature n3 of part_n into feature k2 of part_k
and feature n1 into feature k7).

Subgoal
(double-insert feature n3 of part_n into feature k2 of part_k
and feature n1 into feature k4)
is moved to the list of choice points. There is a functionally symmetrical alternative in
(double-insert feature n3 of part_n into feature k2 of part_k
and feature n1 into feature k7).

Subgoal
(double-insert feature n3 of part_n into feature k2 of part_k
and feature n1 into feature k3)

is moved to the list of choice points. There is a functionally symmetrical alternative in
(double-insert feature n3 of part_n into feature k2 of part_k
and feature n1 into feature k7).

Subgoal Selection:

Feature Level Subgoals	Confidence Rates
(double-insert feature n3 of part_n into feature k5 of part_k and feature n1 into feature k6)	0.22916666
(double-insert feature n3 of part_n into feature k2 of part_k and feature n1 into feature k7)	0.0625

Based on the confidence rates associated with feature level subgoals
(double-insert feature n3 of part_n into feature k5 of part_k
and feature n1 into feature k6)
is chosen as the hypothesized subgoal.

Creating feature level subgoals for object level goal

(double-insert part_m into part_k) .....

.
.
.
.
.

Hypothesized subgoal:

(double-insert feature m3 of part_m into feature k5 of part_k
and feature m1 into feature k6)

Creating feature level subgoals for object level goal

(double-insert part_1 into part_k) .....

After applying the constraints associated with the operations
the feature level subgoals are:

Feature Level Subgoals

(double-insert feature l1 of part_1 into feature k2 of part_k
and feature l3 into feature k7).

(double-insert feature l1 of part_1 into feature k2 of part_k
and feature l3 into feature k6).

(double-insert feature l1 of part_1 into feature k2 of part_k
and feature l3 into feature k5).

(double-insert feature l1 of part_1 into feature k2 of part_k
and feature l3 into feature k4).

(double-insert feature l1 of part_1 into feature k2 of part_k
and feature l3 into feature k3).

Applying beta-symmetry of objects to reduce search tree ..

Applying alpha-symmetry of objects to reduce search tree..

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(double-insert feature l1 of part_1 into feature k2 of part_k and feature l3 into feature k7)	0.2
(double-insert feature l1 of part_1 into feature k2 of part_k and feature l3 into feature k6)	0.2

```
(double-insert feature l1 of part_l into feature k2 of part_k
          and feature l3 into feature k5)                  0.2

(double-insert feature l1 of part_l into feature k2 of part_k
          and feature l3 into feature k4)                  0.2

(double-insert feature l1 of part_l into feature k2 of part_k
          and feature l3 into feature k3)                  0.2
```

Functional Symmetry:

Subgoal
(double-insert feature l1 of part_l into feature k2 of part_k
and feature l3 into feature k6)
is moved to the list of choice points. There is a functionally symmetrical alternative in
(double-insert feature l1 of part_l into feature k2 of part_k
and feature l3 into feature k7).

Subgoal
(double-insert feature l1 of part_l into feature k2 of part_k
and feature l3 into feature k5)
is moved to the list of choice points. There is a functionally symmetrical alternative in
(double-insert feature l1 of part_l into feature k2 of part_k
and feature l3 into feature k7).

Subgoal
(double-insert feature l1 of part_l into feature k2 of part_k
and feature l3 into feature k4)
is moved to the list of choice points. There is a functionally symmetrical alternative in
(double-insert feature l1 of part_l into feature k2 of part_k
and feature l3 into feature k7).

Subgoal
(double-insert feature l1 of part_l into feature k2 of part_k
and feature l3 into feature k3)
is moved to the list of choice points. There is a functionally symmetrical alternative in
(double-insert feature l1 of part_l into feature k2 of part_k
and feature l3 into feature k7).

Hypothesized subgoal:

```
(double-insert feature l1 of part_l into feature k2 of part_k
                              and feature l3 into feature k7).

-----------------------------------------------------
                Accessibility Analysis
-----------------------------------------------------

An alternative feature subgoal
(double-insert feature m3 of part_m into feature k3 of part_k
                              and feature m1 into feature k4)
is chosen for the object level goal
(double-insert part_m into part_k).

-----------------------------------------------------
          Tentative Plan at the Feature Level
-----------------------------------------------------

(double-insert feature n3 of part_n into feature k5 of part_k
                              and feature n1 into feature k6).

(double-insert feature m3 of part_m into feature k3 of part_k
                              and feature m1 into feature k4).

(double-insert feature l1 of part_l into feature k2 of part_k
                              and feature l3 into feature k7).
```

C.3 RAPT Input Generated by ASSEMBLE

As specified in Section 6.6.2, ASSEMBLE generates subfeatures corresponding to the mating features, the requisite information for processing by RAPT including spatial relationships between subfeatures, and produces a file comprising a set of prolog assumptions. This file generated by ASSEMBLE for the Double-insertions experiment is presented below:

```
;;; initialization
```

```
:-init(assume).

;;; declarations pertaining to subfeatures of k5.

:-assume(k5_curved_face$part_k:cylindrical_face).
:-assume(r$k5_curved_face$part_k=0.25).
:-assume(loc$k5_curved_face$part_k=Pos_2),
       Pos_2 is stpos(0,0,1, 1,0,0, 0,1,0, -1.0,2.4142137,0.0).
:-assume(k5_botttom_face$part_k:plane_face).
:-assume(loc$k5_botttom_face$part_k=Pos_3),
       Pos_3 is stpos(0,0,1, 1,0,0, 0,1,0, -1.0,2.4142137,0.0).

;;; declarations pertaining to subfeatures of k6.

:-assume(k6_curved_face$part_k:cylindrical_face).
:-assume(r$k6_curved_face$part_k=0.25).
:-assume(loc$k6_curved_face$part_k=Pos_4),
       Pos_4 is stpos(0,0,1, 1,0,0, 0,1,0, 1.0,2.4142137,0.0).
:-assume(k6_botttom_face$part_k:plane_face).
:-assume(loc$k6_botttom_face$part_k=Pos_5),
       Pos_5 is stpos(0,0,1, 1,0,0, 0,1,0, 1.0,2.4142137,0.0).

;;; declarations pertaining to subfeatures of n3.

:-assume(n3_curved_face$part_n:cylindrical_face).
:-assume(r$n3_curved_face$part_n=0.25).
:-assume(loc$n3_curved_face$part_n=Pos_6),
       Pos_6 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.25,1.0,0).
:-assume(n3_botttom_face$part_n:plane_face).
:-assume(loc$n3_botttom_face$part_n=Pos_7),
       Pos_7 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.25,1.0,0).

;;; declarations pertaining to subfeatures of n1.

:-assume(n1_curved_face$part_n:cylindrical_face).
:-assume(r$n1_curved_face$part_n=0.25).
:-assume(loc$n1_curved_face$part_n=Pos_8),
       Pos_8 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.25,-1.0,0).
:-assume(n1_botttom_face$part_n:plane_face).
:-assume(loc$n1_botttom_face$part_n=Pos_9),
       Pos_9 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.25,-1.0,0).

;;; spatial relations between subfeatures of n3 and k5,
;;; and n1 and k6.
```

```
:-assume(n3_botttom_face$part_n against k5_botttom_face$part_k).
:-assume(n3_curved_face$part_n fits k5_curved_face$part_k).
:-assume(n1_botttom_face$part_n against k6_botttom_face$part_k).
:-assume(n1_curved_face$part_n fits k6_curved_face$part_k).

;;; declarations pertaining to subfeatures of k3.

:-assume(k3_curved_face$part_k:cylindrical_face).
:-assume(r$k3_curved_face$part_k=0.25).
:-assume(loc$k3_curved_face$part_k=Pos_10),
       Pos_10 is stpos(0,0,1, 1,0,0, 0,1,0, -2.4142137,-1.0,0.0).
:-assume(k3_botttom_face$part_k:plane_face).
:-assume(loc$k3_botttom_face$part_k=Pos_11),
       Pos_11 is stpos(0,0,1, 1,0,0, 0,1,0, -2.4142137,-1.0,0.0).

;;; declarations pertaining to subfeatures of k4.

:-assume(k4_curved_face$part_k:cylindrical_face).
:-assume(r$k4_curved_face$part_k=0.25).
:-assume(loc$k4_curved_face$part_k=Pos_12),
       Pos_12 is stpos(0,0,1, 1,0,0, 0,1,0, -2.4142137,1.0,0.0).
:-assume(k4_botttom_face$part_k:plane_face).
:-assume(loc$k4_botttom_face$part_k=Pos_13),
       Pos_13 is stpos(0,0,1, 1,0,0, 0,1,0, -2.4142137,1.0,0.0).

;;; declarations pertaining to subfeatures of m3.

:-assume(m3_curved_face$part_m:cylindrical_face).
:-assume(r$m3_curved_face$part_m=0.25).
:-assume(loc$m3_curved_face$part_m=Pos_14),
       Pos_14 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.25,1.0,0).
:-assume(m3_botttom_face$part_m:plane_face).
:-assume(loc$m3_botttom_face$part_m=Pos_15),
       Pos_15 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.25,1.0,0).

;;; declarations pertaining to subfeatures of m1.

:-assume(m1_curved_face$part_m:cylindrical_face).
:-assume(r$m1_curved_face$part_m=0.25).
:-assume(loc$m1_curved_face$part_m=Pos_16),
       Pos_16 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.25,-1.0,0).
:-assume(m1_botttom_face$part_m:plane_face).
:-assume(loc$m1_botttom_face$part_m=Pos_17),
       Pos_17 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.25,-1.0,0).
```

```
;;; spatial relations between subfeatures of m3 and k3,
;;; and m1 and k4.

:-assume(m3_botttom_face$part_m against k3_botttom_face$part_k).
:-assume(m3_curved_face$part_m fits k3_curved_face$part_k).
:-assume(m1_botttom_face$part_m against k4_botttom_face$part_k).
:-assume(m1_curved_face$part_m fits k4_curved_face$part_k).

;;; declarations pertaining to subfeatures of k2.

:-assume(k2_curved_face$part_k:cylindrical_face).
:-assume(r$k2_curved_face$part_k=1.0).
:-assume(loc$k2_curved_face$part_k=Pos_18),
        Pos_18 is stpos(0,0,1, 1,0,0, 0,1,0, 0.0,0.0,0.0).
:-assume(k2_botttom_face$part_k:plane_face).
:-assume(loc$k2_botttom_face$part_k=Pos_19),
        Pos_19 is stpos(0,0,1, 1,0,0, 0,1,0, 0.0,0.0,0.0).

;;; declarations pertaining to subfeatures of k7.

:-assume(k7_curved_face$part_k:cylindrical_face).
:-assume(r$k7_curved_face$part_k=0.25).
:-assume(loc$k7_curved_face$part_k=Pos_20),
  Pos_20 is stpos(0,0,1, 1,0,0, 0,1,0, 1.847759,-1.847759,0.0).
:-assume(k7_botttom_face$part_k:plane_face).
:-assume(loc$k7_botttom_face$part_k=Pos_21),
  Pos_21 is stpos(0,0,1, 1,0,0, 0,1,0, 1.847759,-1.847759,0.0).

;;; declarations pertaining to subfeatures of l1.

:-assume(l1_curved_face$part_l:cylindrical_face).
:-assume(r$l1_curved_face$part_l=1.0).
:-assume(loc$l1_curved_face$part_l=Pos_22),
        Pos_22 is stpos(0,0,-1, 1,0,0, 0,-1,0, -1.6815629,0,-2.25).
:-assume(l1_botttom_face$part_l:plane_face).
:-assume(loc$l1_botttom_face$part_l=Pos_23),
        Pos_23 is stpos(0,0,-1, 1,0,0, 0,-1,0, -1.6815629,0,-2.25).

;;; declarations pertaining to subfeatures of l3.

:-assume(l3_curved_face$part_l:cylindrical_face).
:-assume(r$l3_curved_face$part_l=0.25).
:-assume(loc$l3_curved_face$part_l=Pos_24),
        Pos_24 is stpos(0,0,-1, 1,0,0, 0,-1,0, 0.9315629,0,-2.25).
:-assume(l3_botttom_face$part_l:plane_face).
```

```
:-assume(loc$l3_botttom_face$part_l=Pos_25),
        Pos_25 is stpos(0,0,-1, 1,0,0, 0,-1,0, 0.9315629,0,-2.25).

;;; spatial relations between subfeatures of l1 and k2,
;;; and l3 and k7.

:-assume(l1_botttom_face$part_l against k2_botttom_face$part_k).
:-assume(l1_curved_face$part_l fits k2_curved_face$part_k).
:-assume(l3_botttom_face$part_l against k7_botttom_face$part_k).
:-assume(l3_curved_face$part_l fits k7_curved_face$part_k).

;;; absolute location of part_k.

:-assume(loc$part_k=trans(0,0,0)).
```

C.4 Output Produced by RAPT

The output generated by RAPT, indicating the locations of part_l, part_m, and part_n, based on the location specified for part_k, is given below:

```
The following relationships and locations will be used:
l1_botttom_face $ part_l against k2_botttom_face $ part_k
l3_botttom_face $ part_l against k7_botttom_face $ part_k
m1_botttom_face $ part_m against k4_botttom_face $ part_k
m3_botttom_face $ part_m against k3_botttom_face $ part_k
n1_botttom_face $ part_n against k6_botttom_face $ part_k
n3_botttom_face $ part_n against k5_botttom_face $ part_k
l1_curved_face $ part_l fits k2_curved_face $ part_k
l3_curved_face $ part_l fits k7_curved_face $ part_k
m1_curved_face $ part_m fits k4_curved_face $ part_k
m3_curved_face $ part_m fits k3_curved_face $ part_k
n1_curved_face $ part_n fits k6_curved_face $ part_k
n3_curved_face $ part_n fits k5_curved_face $ part_k
loc$part_k = pos(1,0,0; 0,1,0; 0,0,1; 0,0,0)

 Message from RAPT:


The following relationships and locations will be used:
l1_botttom_face $ part_l against k2_botttom_face $ part_k
l3_botttom_face $ part_l against k7_botttom_face $ part_k
```

```
m1_botttom_face $ part_m against k4_botttom_face $ part_k
m3_botttom_face $ part_m against k3_botttom_face $ part_k
n1_botttom_face $ part_n against k6_botttom_face $ part_k
n3_botttom_face $ part_n against k5_botttom_face $ part_k
l1_curved_face $ part_l fits k2_curved_face $ part_k
l3_curved_face $ part_l fits k7_curved_face $ part_k
m1_curved_face $ part_m fits k4_curved_face $ part_k
m3_curved_face $ part_m fits k3_curved_face $ part_k
n1_curved_face $ part_n fits k6_curved_face $ part_k
n3_curved_face $ part_n fits k5_curved_face $ part_k


The following locations have been inferred:

loc $ part_l = pos(0.707107,-0.707107,0.0; 0.707107,0.707107,0.0;
                   0.0,0.0,1.0; 1.18904,-1.18904,2.25)
loc $ part_k = pos(1.0,0.0,0.0; 0.0,1.0,0.0;
                   0.0,0.0,1.0; 0.0,0.0,0.0)
loc $ part_m = pos(0.0,0.0,1.0; -0.0,-1.0,0.0;
                   1.0,0.0,0.0; -2.41421,-0.000001,2.25)
loc $ part_n = pos(0.0,0.0,1.0; -1.0,-0.0,0.0;
                   -0.0,-1.0,0.0; 0.000001,2.41421,2.25)
```

The values of variables loc$part_l, loc$part_m, and loc$part_n give the locations inferred by RAPT for objects part_l, part_m and part_n, respectively.

APPENDIX D

THE LIGHT BULB FIXTURE ASSEMBLY EXPERIMENT

This appendix provides the input to and output from the ASSEMBLE and RAPT systems for the Light bulb fixture assembly experiment described in Section 7.4. The appendix contains four sections: the input provided to ASSEMBLE, the trace ASSEMBLE produces in the course of executing the FIND-FEATURES subsystem and the accessibility analysis phase of the VERIFY subsystem, the file containing the prolog declarations that constitute the input to RAPT, and finally the output produced by RAPT.

D.1 Input to ASSEMBLE

The following constitutes the input to ASSEMBLE corresponding to the Light Bulb Fixture assembly experiment:

```
;;; load file containing object and feature information
(load "lbfixture.lisp")

;;; input assembly instructions
(insert part_b in part_a)
(thread part_a with part_c)
(thread part_b with part_d)

;;; precedence relations
(2 1) (3 2)

;;; name of file to contain input to RAPT
"lbfixture.pl"

;;; object relative to which locations of other objects are deduced
```

```
part_a
```

D.1.1 Object and Feature Information

The structures generated for the objects part_a, part_b, part_c, and part_d and their features from reading the input file `lbfixture.lisp` are shown below.

Input information pertaining to part_a:

```
#<object 1546741> is a structure of type object
   NAME:                          part_a
   SHAPE:                         cylinder
   LENGTH:                        3.6
   WIDTH:                         3.6
   HEIGHT:                        2.4
   ALPHA:                         #(0 1 0 6.283185307179587d0)
   BETA:                          #(0 0 1 3.141592653589793d0)
   FEATURES:                      (a1 a2 a3 a4)
```

The following structures correspond to the features of part_a:

```
#<feature 1547046> is a structure of type feature
   FEATURE-NAME:                  a1
   FEATURE-FUNCTION:              container
   FEATURE-SHAPE:                 cylindrical-hole
   FEATURE-DIMENSIONS:            (1.0 2.4)
   FEATURE-POSITION:              #2A((0 0 -1.2 1.0) (0 0 -1 0))
   FEATURE-BETA:                  0.0
   ADJACENT-FEATURES:             (a2)
   COAXIAL-FEATURES:              (a2)

#<feature 1547155> is a structure of type feature
   FEATURE-NAME:                  a2
   FEATURE-FUNCTION:              container
   FEATURE-SHAPE:                 thread
   FEATURE-DIMENSIONS:            (1.0 0.5 0.1 t)
   FEATURE-POSITION:              #2A((0 0 -1.2 1.0) (0 0 -1 0))
   FEATURE-BETA:                  0.0
   ADJACENT-FEATURES:             (a1)
   COAXIAL-FEATURES:              (a1)
```

```
#<feature 1547347> is a structure of type feature
   FEATURE-NAME:                a3
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               cylindrical-hole
   FEATURE-DIMENSIONS:          (3.6 0.7)
   FEATURE-POSITION:            #2A((0 0 1.2 1.0) (0 0 1 0))
   FEATURE-BETA:                3.14159265358979360
   ADJACENT-FEATURES:           (a4)
   COAXIAL-FEATURES:            (a4)

#<feature 1547456> is a structure of type feature
   FEATURE-NAME:                a4
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               thread
   FEATURE-DIMENSIONS:          (3.6 0.5 0.25 nil)
   FEATURE-POSITION:            #2A((0 0 1.2 1.0) (0 0 1 0))
   FEATURE-BETA:                0.0
   ADJACENT-FEATURES:           (a3)
   COAXIAL-FEATURES:            (a3)
```

Input information pertaining to part_b:

```
#<object 1547574> is a structure of type object
   NAME:                        part_b
   SHAPE:                       box-plus-cylinder
   LENGTH:                      3.2
   WIDTH:                       3.1
   HEIGHT:                      2.9
   ALPHA:                       #(1 0 0 6.283185307179587d0)
   BETA:                        #(0 1 0 3.14159265358979360)
   FEATURES:                    (b1 b2 b3 b4)
```

Features of part_b:

```
#<feature 1547701> is a structure of type feature
   FEATURE-NAME:                b1
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               rectangular-face
   FEATURE-DIMENSIONS:          (3.2 1.6)
   FEATURE-POSITION:            #2A((0 -1.6 0 1.0) (0 -1 0 0))
   FEATURE-BETA:                3.14159265358979360

#<feature 1550006> is a structure of type feature
```

```
    FEATURE-NAME:                 b2
    FEATURE-FUNCTION:             container
    FEATURE-SHAPE:                cylindrical-hole
    FEATURE-DIMENSIONS:           (2.9 2.0)
    FEATURE-POSITION:             #2A((0 1.5 0 1.0) (0 1 0 0))
    FEATURE-BETA:                 0.0
    ADJACENT-FEATURES:            (b3)
    COAXIAL-FEATURES:             (b3)

#<feature 1550115> is a structure of type feature
    FEATURE-NAME:                 b3
    FEATURE-FUNCTION:             insertor
    FEATURE-SHAPE:                thread
    FEATURE-DIMENSIONS:           (2.9 1.3 0.4 nil)
    FEATURE-POSITION:             #2A((0 1.5 0 1.0) (0 1 0 0))
    FEATURE-BETA:                 0.0
    ADJACENT-FEATURES:            (b2 b4)
    COAXIAL-FEATURES:             (b2)

#<feature 1550307> is a structure of type feature
    FEATURE-NAME:                 b4
    FEATURE-FUNCTION:             insertor
    FEATURE-SHAPE:                cylindrical-surface
    FEATURE-DIMENSIONS:           (2.6 3.1)
    FEATURE-POSITION:             #2A((0 0 1.45 1.0) (0 0 1 0))
    FEATURE-BETA:                 6.283185307179587d0
    ADJACENT-FEATURES:            (b3)
```

Input information pertaining to part_c:

```
#<object 1550432> is a structure of type object
    NAME:                         part_c
    SHAPE:                        cylinder
    LENGTH:                       3.6
    WIDTH:                        3.6
    HEIGHT:                       2.1
    ALPHA:                        #(0 1 0 6.2831853071795870)
    BETA:                         #(0 0 1 0.0)
    FEATURES:                     (c1 c2 c3)
```

Features of part_c:

```
#<feature 1550537> is a structure of type feature
```

```
    FEATURE-NAME:                c1
    FEATURE-FUNCTION:            container
    FEATURE-SHAPE:               cylindrical-hole
    FEATURE-DIMENSIONS:          (3.6 2.1)
    FEATURE-POSITION:            #2A((0 0 -1.1 1.0) (0 0 -1 0))
    FEATURE-BETA:                0.0
    ADJACENT-FEATURES:           (c2)
    COAXIAL-FEATURES:            (c2)

#<feature 1550646> is a structure of type feature
    FEATURE-NAME:                c2
    FEATURE-FUNCTION:            container
    FEATURE-SHAPE:               thread
    FEATURE-DIMENSIONS:          (3.6 0.5 0.25 t)
    FEATURE-POSITION:            #2A((0 0 -1.1 1.0) (0 0 -1 0))
    FEATURE-BETA:                0.0
    ADJACENT-FEATURES:           (c1)
    COAXIAL-FEATURES:            (c1)

#<feature 1550753> is a structure of type feature
    FEATURE-NAME:                c3
    FEATURE-FUNCTION:            container
    FEATURE-SHAPE:               cylindrical-hole
    FEATURE-DIMENSIONS:          (3.3 2.1)
    FEATURE-POSITION:            #2A((0 0 1.0 1.0) (0 0 1 0))
    FEATURE-BETA:                0.0
```

Input information pertaining to part_d:

```
#<object 1551073> is a structure of type object
    NAME:                        part_d
    SHAPE:                       cylinder
    LENGTH:                      3.5
    WIDTH:                       3.5
    HEIGHT:                      1.4
    ALPHA:                       #(0 1 0 6.283185307179587d0)
    BETA:                        #(0 0 1 0.0)
    FEATURES:                    (d1 d2 d3 d4)
```

Features of part_d:

```
#<feature 1551264> is a structure of type feature
    FEATURE-NAME:                d1
```

```
    FEATURE-FUNCTION:           container
    FEATURE-SHAPE:              cylindrical-hole
    FEATURE-DIMENSIONS:         (2.9 1.4)
    FEATURE-POSITION:           #2A((0 0 0.7 1.0) (0 0 1 0))
    FEATURE-BETA:               0.0
    ADJACENT-FEATURES:          (d2)
    OVERLAPPING-FEATURES:       (d3)
    COAXIAL-FEATURES:           (d2)

#<feature 1551373> is a structure of type feature
    FEATURE-NAME:               d2
    FEATURE-FUNCTION:           container
    FEATURE-SHAPE:              thread
    FEATURE-DIMENSIONS:         (2.9 1.3 0.4 t)
    FEATURE-POSITION:           #2A((0 0 0.7 1.0) (0 0 1 0))
    FEATURE-BETA:               0.0
    ADJACENT-FEATURES:          (d1)
    OVERLAPPING-FEATURES:       (d4)
    COAXIAL-FEATURES:           (d1)

#<feature 1551500> is a structure of type feature
    FEATURE-NAME:               d3
    FEATURE-FUNCTION:           container
    FEATURE-SHAPE:              cylindrical-hole
    FEATURE-DIMENSIONS:         (2.9 1.4)
    FEATURE-POSITION:           #2A((0 0 -0.7 1.0) (0 0 -1 0))
    FEATURE-BETA:               0.0
    ADJACENT-FEATURES:          (d4)
    OVERLAPPING-FEATURES:       (d1)
    COAXIAL-FEATURES:           (d4)

#<feature 1551607> is a structure of type feature
    FEATURE-NAME:               d4
    FEATURE-FUNCTION:           container
    FEATURE-SHAPE:              thread
    FEATURE-DIMENSIONS:         (2.9 1.3 0.4 t)
    FEATURE-POSITION:           #2A((0 0 -0.7 1.0) (0 0 -1 0))
    FEATURE-BETA:               0.0
    ADJACENT-FEATURES:          (d3)
    OVERLAPPING-FEATURES:       (d2)
    COAXIAL-FEATURES:           (d3)
```

D.1.2 Addition of Secondary Features

In applying the distinguishing-features heuristic the features a5 and a6 are added to the representation of object part_a. The information about these features coded in the system is given below:

```
#<feature 7247443> is a structure of type feature
   FEATURE-NAME:                a5
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               semi-cylindrical-depression
   FEATURE-DIMENSIONS:          (0.2 0.1)
   FEATURE-POSITION:            #2A((0 0 0 1) (0 0 1 0))
   ADJACENT-FEATURES:           (a4)
   ENCOMPASSING-FEATURE:        a3

#<feature 7247464> is a structure of type feature
   FEATURE-NAME:                a6
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               semi-cylindrical-depression
   FEATURE-DIMENSIONS:          (0.2 0.1)
   FEATURE-POSITION:            #2A((0 0 0 1) (0 0 1 0))
   ADJACENT-FEATURES:           (a4)
   ENCOMPASSING-FEATURE:        a3
```

The features added to the representation of part_b are b6 and b7. The structures corresponding to these features are given below:

```
#<feature 7247505> is a structure of type feature
   FEATURE-NAME:                b6
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               semi-cylindrical-depression
   FEATURE-DIMENSIONS:          (0.2 0.1)
   FEATURE-POSITION:            #2A((0 0 0 1) (0 0 1 0))
   ENCOMPASSING-FEATURE:        b1

#<feature 7247526> is a structure of type feature
   FEATURE-NAME:                b7
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               semi-cylindrical-depression
   FEATURE-DIMENSIONS:          (0.2 0.1)
   FEATURE-POSITION:            #2A((0 0 0 1) (0 0 1 0))
   ENCOMPASSING-FEATURE:        b1
```

D.2 A Trace of ASSEMBLE's Processing of the Task

In the course of building and modifying the goal network corresponding to this experimental assembly task, ASSEMBLE writes out intermediate states of the goal network. The trace genertated corresponding to the Light bulb Fixture assembly experiment is presented below:

```
--------------------------------------------------
                  Program Input
--------------------------------------------------

List of Goals

1 : (thread part_b with part_d)

2 : (thread part_a with part_c)

3 : (insert part_b in part_a)


Ordering Constraints

Goal 2 precedes Goal 1.
Goal 3 precedes Goal 2.


--------------------------------------------------


Creating feature level subgoals for object level goal

                (thread part_b with part_d) ........


After applying the constraints associated with the operations
the feature level subgoals are:

      Feature Level Subgoals

      (thread feature b3 of part_b with feature d4 of part_d).

      (thread feature b3 of part_b with feature d2 of part_d).
```

Applying beta-symmetry of objects to reduce search tree ..

Applying alpha-symmetry of objects to reduce search tree..

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(thread feature b3 of part_b with feature d4 of part_d)	0.5
(thread feature b3 of part_b with feature d2 of part_d)	0.5

Functional symmetry:

Subgoal (thread feature b3 of part_b with feature d2 of part_d) is moved to the list of choice points. There is a functionally symmetrical alternative in
(thread feature b3 of part_b with feature d4 of part_d).

Hypothesized subgoal:

(thread feature b3 of part_b with feature d4 of part_d).

Creating feature level subgoals for object level goal

(thread part_a with part_c) ....

After applying the constraints associated with the operations the feature level subgoals are:

Feature Level Subgoals

(thread feature a4 of part_a with feature c2 of part_c).

Applying beta-symmetry of objects to reduce search tree ..

Applying alpha-symmetry of objects to reduce search tree..

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(thread feature a4 of part_a with feature c2 of part_c)	1.0

Creating feature level subgoals for object level goal

(insert part_b in part_a) ....

After applying the constraints associated with the operations the feature level subgoals are:

Feature Level Subgoals

(insert feature b4 of part_b in feature a3 of part_a).

(insert feature b2 of part_b in feature a3 of part_a).

(insert feature b1 of part_b in feature a3 of part_a).

Applying beta-symmetry of objects to reduce search tree ..

Applying alpha-symmetry of objects to reduce search tree..

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(insert feature b4 of part_b in feature a3 of part_a)	0.33333334
(insert feature b2 of part_b in feature a3 of part_a)	0.33333334
(insert feature b1 of part_b in feature a3 of part_a)	0.33333334

Heuristic closeness-of-fit modifies the confidence rates of subgoals as follows:

(insert feature b4 of part_b in feature a3 of part_a)	0.16666667
(insert feature b2 of part_b in feature a3 of part_a)	0.16666667
(insert feature b1 of part_b in feature a3 of part_a)	0.6666667

Heuristic find-distinguishing-feature applies to feature level subgoal:

(insert feature b1 of part_b in feature a3 of part_a).

The confidence rates of all the feature-level subgoals of (insert part_b in part_a) are modified as follows:

(insert feature b4 of part_b in feature a3 of part_a)	0.083333336
(insert feature b2 of part_b in feature a3 of part_a)	0.083333336
(insert feature b1 of part_b in feature a3 of part_a)	0.8333334

Subgoal Selection:

Feature Level Subgoals	Confidence Rates

```
(insert feature b4 of part_b in feature a3 of part_a)  0.083333336

(insert feature b2 of part_b in feature a3 of part_a)  0.083333336

(insert feature b1 of part_b in feature a3 of part_a)   0.8333334

Based on the confidence rates associated with feature level
subgoals
(insert feature b1 of part_b in feature a3 of part_a)
has been chosen as the hypothesized subgoal.


--------------------------------------------------
                Accessibility Analysis
--------------------------------------------------



--------------------------------------------------
          Tentative Plan at the Feature Level
--------------------------------------------------

 (thread feature b3 of part_b with feature d4 of part_d).

 (thread feature a4 of part_a with feature c2 of part_c).

 (insert feature b1 of part_b in feature a3 of part_a).
```

D.3 RAPT Input Generated by ASSEMBLE

As specified in Section 6.6.2, ASSEMBLE generates subfeatures corresponding to the mating features, the requisite information for processing by RAPT including spatial relationships between subfeatures, and produces a file comprising a set of prolog assumptions. This file generated by ASSEMBLE for the Light bulb fixture assembly experiment is presented below:

```
;;; initialization
:-init(assume).
```

```
;;; declarations pertaining to subfeatures of d4.

:-assume(d4_curved_face$part_d:cylindrical_face).
:-assume(r$d4_curved_face$part_d=1.45).
:-assume(loc$d4_curved_face$part_d=Pos_4),
       Pos_4 is stpos(0,0,-1, 1,0,0, 0,-1,0, 0.0,0.0,0.59999996).
:-assume(d4_botttom_face$part_d:plane_face).
:-assume(loc$d4_botttom_face$part_d=Pos_5),
       Pos_5 is stpos(0,0,-1, 1,0,0, 0,-1,0, 0.0,0.0,0.59999996).

;;; declarations pertaining to subfeatures of b3.

:-assume(b3_curved_face$part_b:cylindrical_face).
:-assume(r$b3_curved_face$part_b=1.45).
:-assume(loc$b3_curved_face$part_b=Pos_6),
       Pos_6 is stpos(0,1,0, 1,0,0, 0,0,-1, 0,1.5,0).
:-assume(b3_botttom_face$part_b:plane_face).
:-assume(loc$b3_botttom_face$part_b=Pos_7),
       Pos_7 is stpos(0,1,0, 1,0,0, 0,0,-1, 0,1.5,0).

;;; spatial relations between subfeatures of d4 and b3.

:-assume(b3_botttom_face$part_b against d4_botttom_face$part_d).
:-assume(b3_curved_face$part_b fits d4_curved_face$part_d).

;;; declarations pertaining to subfeatures of c2.

:-assume(c2_curved_face$part_c:cylindrical_face).
:-assume(r$c2_curved_face$part_c=1.8).
:-assume(loc$c2_curved_face$part_c=Pos_10),
   Pos_10 is stpos(0,0,-1, 1,0,0, 0,-1,0, 0.0,0.0,-0.60000005).
:-assume(c2_botttom_face$part_c:plane_face).
:-assume(loc$c2_botttom_face$part_c=Pos_11),
   Pos_11 is stpos(0,0,-1, 1,0,0, 0,-1,0, 0.0,0.0,-0.60000005).

;;; declarations pertaining to subfeatures of a4.

:-assume(a4_curved_face$part_a:cylindrical_face).
:-assume(r$a4_curved_face$part_a=1.8).
:-assume(loc$a4_curved_face$part_a=Pos_12),
       Pos_12 is stpos(0,0,1, 1,0,0, 0,1,0, 0,0,1.2).
:-assume(a4_botttom_face$part_a:plane_face).
:-assume(loc$a4_botttom_face$part_a=Pos_13),
       Pos_13 is stpos(0,0,1, 1,0,0, 0,1,0, 0,0,1.2).
```

```
;;; spatial relations between subfeatures of c2 and a4.

:-assume(a4_botttom_face$part_a against c2_botttom_face$part_c).
:-assume(a4_curved_face$part_a fits c2_curved_face$part_c).

;;; declarations pertaining to subfeatures of a3.

:-assume(a3_curved_face$part_a:cylindrical_face).
:-assume(r$a3_curved_face$part_a=1.8).
:-assume(loc$a3_curved_face$part_a=Pos_14),
        Pos_14 is stpos(0,0,1, 1,0,0, 0,1,0, 0.0,0.0,0.50000006).
:-assume(a3_botttom_face$part_a:plane_face).
:-assume(loc$a3_botttom_face$part_a=Pos_15),
        Pos_15 is stpos(0,0,1, 1,0,0, 0,1,0, 0.0,0.0,0.50000006).

;;; declarations pertaining to subfeatures of b1.

:-assume(b1_curved_face$part_b:cylindrical_face).
:-assume(r$b1_curved_face$part_b=1.7888545).
:-assume(loc$b1_curved_face$part_b=Pos_16),
        Pos_16 is stpos(0,-1,0, 1,0,0, 0,0,1, 0,-1.6,0).
:-assume(b1_botttom_face$part_b:plane_face).
:-assume(loc$b1_botttom_face$part_b=Pos_17),
        Pos_17 is stpos(0,-1,0, 1,0,0, 0,0,1, 0,-1.6,0).

;;; spatial relations between subfeatures of a3 and b1.

:-assume(b1_botttom_face$part_b against a3_botttom_face$part_a).
:-assume(b1_curved_face$part_b fits a3_curved_face$part_a).

;;; absolute location of part_a.

:-assume(loc$part_a=trans(0,0,0)).
```

D.4 Output Produced by RAPT

The output generated by RAPT, indicating the locations of part_b, part_c, and part_d, based on the specified absolute location of part_a is given below:

```
The following relationships and locations will be used:
```

```
a4_bottom_face $ part_a against c2_bottom_face $ part_c
b1_botttom_face $ part_b against a3_botttom_face $ part_a
b3_bottom_face $ part_b against d4_bottom_face $ part_d
a4_curved_face $ part_a fits c2_curved_face $ part_c
b1_curved_face $ part_b fits a3_curved_face $ part_a
b3_curved_face $ part_b fits d4_curved_face $ part_d
loc$part_a = pos(1,0,0; 0,1,0; 0,0,1; 0,0,0)

 Message from RAPT:


The following relationships and locations will be used:
a4_bottom_face $ part_a against c2_bottom_face $ part_c
b1_botttom_face $ part_b against a3_botttom_face $ part_a
b3_bottom_face $ part_b against d4_bottom_face $ part_d
a4_curved_face $ part_a fits c2_curved_face $ part_c
b1_curved_face $ part_b fits a3_curved_face $ part_a
b3_curved_face $ part_b fits d4_curved_face $ part_d


The following locations have been inferred:


 Message from RAPT:


The following relationships and locations will be used:
a4_bottom_face $ part_a against c2_bottom_face $ part_c
b1_botttom_face $ part_b against a3_botttom_face $ part_a
b3_bottom_face $ part_b against d4_bottom_face $ part_d
a4_curved_face $ part_a fits c2_curved_face $ part_c
b1_curved_face $ part_b fits a3_curved_face $ part_a
b3_curved_face $ part_b fits d4_curved_face $ part_d

 Message from RAPT:


The following relationships and locations will be used:
a4_bottom_face $ part_a against c2_bottom_face $ part_c
b1_botttom_face $ part_b against a3_botttom_face $ part_a
b3_bottom_face $ part_b against d4_bottom_face $ part_d
a4_curved_face $ part_a fits c2_curved_face $ part_c
b1_curved_face $ part_b fits a3_curved_face $ part_a
b3_curved_face $ part_b fits d4_curved_face $ part_d
```

```
Message from RAPT:


The following relationships and locations will be used:
a4_bottom_face $ part_a against c2_bottom_face $ part_c
b1_botttom_face $ part_b against a3_botttom_face $ part_a
b3_bottom_face $ part_b against d4_bottom_face $ part_d
a4_curved_face $ part_a fits c2_curved_face $ part_c
b1_curved_face $ part_b fits a3_curved_face $ part_a
b3_curved_face $ part_b fits d4_curved_face $ part_d
loc $ part_a = pos(1,0,0; 0,1,0; 0,0,1; 0,0,0)
loc $ part_c = pos(1,0,0; 0,1,0; 0,0,1; 0.0,0.0,1.8)
loc $ part_b = pos(0,1,0; 0,0,1; 1,0,0; 0.0,0.0,2.1)
loc $ part_d = pos(1,0,0; 0,1,0; 0,0,1; 0.0,0.0,3.0)
```

The values of variables `loc$part_b`, `loc$part_c`, and `loc$part_d` give the locations inferred by RAPT for objects part_b, part_c and part_d, respectively.

APPENDIX E

THE NC-100 CONTROLLER ASSEMBLY

This appendix provides the input to and output from the ASSEMBLE and RAPT systems for the NC-100 controller assembly experiment described in Section 7.5. The appendix contains four sections: the input provided to ASSEMBLE, portions of the trace ASSEMBLE produces in the course of executing the FIND-FEATURES subsystem and the accessibility analysis phase of the VERIFY subsystem, the file containing the prolog declarations that constitute the input to RAPT, and finally the output produced by RAPT.

E.1 Input to ASSEMBLE

The following constitutes the input specification to ASSEMBLE corresponding to the NC-100 Controller assembly experiment:

```
;;; load file containing object and feature information
(load "controller.lisp")

;;;input assembly instructions
(place transformer on base_plate)

(insert s_bolt1 into transformer)
(insert s_bolt2 into transformer)
(insert s_bolt3 into transformer)
(insert s_bolt4 into transformer)

(thread s_bolt1 into base_plate)
(thread s_bolt2 into base_plate)
(thread s_bolt3 into base_plate)
```

```
(thread s_bolt4 into base_plate)

(insert capacitor into base_plate)

(place side_plate_1 on base_plate)
(place side_plate_2 on base_plate)

(place top_plate on side_plate_1)
(place top_plate on side_plate_2)

(insert l_bolt1 into top_plate)
(insert l_bolt2 into top_plate)
(insert l_bolt3 into top_plate)
(insert l_bolt4 into top_plate)

(insert l_bolt1 into side_plate_1)
(insert l_bolt2 into side_plate_1)
(insert l_bolt3 into side_plate_2)
(insert l_bolt4 into side_plate_2)

(thread l_bolt1 into base_plate)
(thread l_bolt2 into base_plate)
(thread l_bolt3 into base_plate)
(thread l_bolt4 into base_plate)

;;; precedence relations
(26 25) (25 24) (24 23) (23 22) (22 21) (21 20)
(20 19) (19 18) (18 17) (17 16) (16 15) (15 14)
(14 13) (13 12) (12 11) (11 10) (10 9) (9 8)
(8 7) (7 6) (6 5) (5 4) (4 3) (3 2) (2 1)

;;; name of file to contain input to RAPT
"controller.pl"

;;; object relative to which locations of other objects are deduced
base_plate
```

E.1.1 Object and Feature Information

The structures generated for the base_plate, transformer, capacitor, side_plates, top_plate, large bolts and small bolts and the features of these objects from reading the input file `controller.lisp` are shown below. As the number of objects and features in this ex-

periment are large, the representations corresponding to only one of the objects from each of the sets of large bolts, small bolts, and side_plates are presented below. The representations for other members of the set are identical.

Input information pertaining to object base_plate:

```
#<object 1573010> is a structure of type object
   NAME:                          base_plate
   SHAPE:                         rectangular-box
   LENGTH:                        18.6
   WIDTH:                         15.0
   HEIGHT:                        2.0
   ALPHA:                         #(0 1 0 6.28318530717958 7d0)
   BETA:                          #(0 0 1 6.28318530717958 7d0)
   FEATURES:                      (plate_top hole lh1 lh2 lh3 lh4
                                              sh1 sh2 sh3 sh4)
```

Input information pertaining to the features of base_plate are given below. As the large holes lh1, lh2, lh3 and lh4 are identical, the attributes of these features vary only in the feature-name and feature-position slots. So, all of the input information corresponding to feature lh1 are presented and only the differing information is presented for features lh2, lh3 and lh4. A similar procedure is adopted in presenting input information pertaining to the smaller holes sh1, sh2, sh3 and sh4.

```
#<feature 1573112> is a structure of type feature
   FEATURE-NAME:                  plate_top
   FEATURE-FUNCTION:              insertor
   FEATURE-SHAPE:                 rectangular-face
   FEATURE-DIMENSIONS:            (18.6 15.0)
   FEATURE-POSITION:              #2A((0 0 1 1) (0 0 1 0))
   FEATURE-BETA:                  360
   INCLUSIVE-FEATURES:            (hole lh1 lh2 lh3 lh4
                                        sh1 sh2 sh3 sh4)

#<feature 17011432> is a structure of type feature
   FEATURE-NAME:                  hole
   FEATURE-FUNCTION:              container
   FEATURE-SHAPE:                 cylindrical-hole
   FEATURE-DIMENSIONS:            (5.6 2.0)
   FEATURE-POSITION:              #2A((4.95 1.15 1 1) (0 0 1 0))
   FEATURE-BETA:                  0.0
```

```
    ENCOMPASSING-FEATURE:          plate_top

#<feature 1573412> is a structure of type feature
    FEATURE-NAME:                  lh1
    FEATURE-FUNCTION:              container
    FEATURE-SHAPE:                 thread
    FEATURE-DIMENSIONS:            (1.2 2.0 0.1 t)
    FEATURE-POSITION:              #2A((5.6 6.8 1 1) (0 0 1 0))
    FEATURE-BETA:                  0.0
    ENCOMPASSING-FEATURE:          plate_top

#<feature 1573521> is a structure of type feature
    FEATURE-NAME:                  lh2
    FEATURE-POSITION:              #2A((5.6 -6.8 1 1) (0 0 1 0))

#<feature 1573630> is a structure of type feature
    FEATURE-NAME:                  lh3
    FEATURE-POSITION:              #2A((-5.6 6.8 1 1) (0 0 1 0))

#<feature 1573737> is a structure of type feature
    FEATURE-NAME:                  lh4
    FEATURE-POSITION:              #2A((-5.6 -6.8 1 1) (0 0 1 0))

#<feature 1574046> is a structure of type feature
    FEATURE-NAME:                  sh1
    FEATURE-FUNCTION:              container
    FEATURE-SHAPE:                 thread
    FEATURE-DIMENSIONS:            (0.5 0.3 0.05 t)
    FEATURE-POSITION:              #2A((-8.1 -4.8 1 1) (0 0 1 0))
    FEATURE-BETA:                  0.0
    ENCOMPASSING-FEATURE:          plate_top

#<feature 1574155> is a structure of type feature
    FEATURE-NAME:                  sh2
    FEATURE-POSITION:              #2A((-8.1 4.8 1 1) (0 0 1 0))

#<feature 1574360> is a structure of type feature
    FEATURE-NAME:                  sh3
    FEATURE-POSITION:              #2A((-2.4 -4.8 1 1) (0 0 1 0))

#<feature 1574467> is a structure of type feature
    FEATURE-NAME:                  sh4
    FEATURE-POSITION:              #2A((-2.4 4.8 1 1) (0 0 1 0))
```

Information pertaining to object transformer:

```
#<object 1574605> is a structure of type object
   NAME:                          transformer
   SHAPE:                         rectangular-box
   LENGTH:                        11.4
   WIDTH:                         9.8
   HEIGHT:                        6.1
   ALPHA:                         #(0 1 0 6.283185307179587d0)
   BETA:                          #(1 0 0 3.14159265358979360)
   FEATURES:                      (top bottom h1 h2 h3 h4)
```

Input information pertaining to features of the transformer are presented below. As with the case of the base_plate, the holes h1, h2, h3 and h4 have identical attributes except for their positions in the object coordinate frame. In the following, the input information corresponding to feature h1 is provided in full, and only the position attribute is specified for features h2, h3 and h4. It is implicit that values of other attributes of these features are the same as the corresponding attributes of h1.

```
#<feature 1574710> is a structure of type feature
   FEATURE-NAME:                  top
   FEATURE-FUNCTION:              insertor
   FEATURE-SHAPE:                 rectangular-face
   FEATURE-DIMENSIONS:            (9.8 6.1)
   FEATURE-POSITION:              #2A((5.2 0 0 1) (1 0 0 0))
   FEATURE-BETA:                  3.14159265358979360

#<feature 1575013> is a structure of type feature
   FEATURE-NAME:                  bottom
   FEATURE-FUNCTION:              insertor
   FEATURE-SHAPE:                 rectangular-face
   FEATURE-DIMENSIONS:            (9.8 6.1)
   FEATURE-POSITION:              #2A((-5.2 0 0 1) (-1 0 0 0))
   FEATURE-BETA:                  3.14159265358979360

#<feature 1575120> is a structure of type feature
   FEATURE-NAME:                  h1
   FEATURE-FUNCTION:              container
   FEATURE-SHAPE:                 cylindrical-hole
   FEATURE-DIMENSIONS:            (0.5 0.2)
```

```
   FEATURE-POSITION:              #2A((-5 -4.8 -2.85 1) (1 0 0 0))
   FEATURE-BETA:                  0.0

#<feature 1575225> is a structure of type feature
   FEATURE-NAME:                  h2
   FEATURE-POSITION:              #2A((-5 4.8 -2.85 1) (1 0 0 0))

#<feature 1575332> is a structure of type feature
   FEATURE-NAME:                  h3
   FEATURE-POSITION:              #2A((-5 4.8 2.85 1) (1 0 0 0))

#<feature 1575525> is a structure of type feature
   FEATURE-NAME:                  h4
   FEATURE-POSITION:              #2A((-5 -4.8 2.85 1) (1 0 0 0))
```

Information pertaining to object Capacitor:

```
#<object 1575643> is a structure of type object
   NAME:                          capacitor
   SHAPE:                         cylinder
   LENGTH:                        11.2
   WIDTH:                         7.0
   HEIGHT:                        7.0
   ALPHA:                         #(0 1 0 6.283185307179587d0)
   BETA:                          #(1 0 0 3.141592653589793d0)
   FEATURES:                      (c_top c_side)
```

Features of capacitor:

```
#<feature 1575745> is a structure of type feature
   FEATURE-NAME:                  c_top
   FEATURE-FUNCTION:              insertor
   FEATURE-SHAPE:                 circular-face
   FEATURE-DIMENSIONS:            (5.0)
   FEATURE-POSITION:              #2A((5.6 0 0 1) (1 0 0 0))
   FEATURE-BETA:                  180
   ADJACENT-FEATURES:             (c_side)

#<feature 1576050> is a structure of type feature
   FEATURE-NAME:                  c_side
   FEATURE-FUNCTION:              insertor
   FEATURE-SHAPE:                 cylindrical-shaft
   FEATURE-DIMENSIONS:            (5.0 11.2)
```

```
   FEATURE-POSITION:          #2A((-5.6 0 0 1) (-1 0 0 0))
   FEATURE-BETA:              0.0
   ADJACENT-FEATURES:         (c_top)
```

Information pertaining to the side_plates:

```
#<object 1576166> is a structure of type object
   NAME:                      side_plate_1
   SHAPE:                     rectangular-box
   LENGTH:                    18.6
   WIDTH:                     10.2
   HEIGHT:                    1.7
   ALPHA:                     #(0 0 1 3.141592653589793d0)
   BETA:                      #(0 1 0 6.283185307179587d0)
   FEATURES:                  (sp1_face1 sp1_face2 sp1_face3
                               sp1_h1 sp1_h2 sp1_h3 sp1_h4)
```

Input information pertaining to the features of side_plate_1 are presented below. The rectangular faces sp1_face2 and sp1_face3 are identical in shape and dimensions and differ only in their position and the features they include. Similarly, the holes on the side_plate sp1_h1, sp1_h2, sp1_h3, and sp1_h4 have identical attributes except for their position and feature relationships. So, the input information for sp1_face2 and sp1_h1 are provided in full, and for the other features listed above, only the differing attributes are presented.

```
#<feature 1576271> is a structure of type feature
   FEATURE-NAME:              sp1_face1
   FEATURE-FUNCTION:          insertor
   FEATURE-SHAPE:             rectangular-face
   FEATURE-DIMENSIONS:        (18.6 10.2)
   FEATURE-POSITION:          #2A((0 0 0.85 1) (0 0 1 0))
   FEATURE-BETA:              3.141592653589793d0
   ADJACENT-FEATURES:         (sp1_face2 sp1_face3)

#<feature 1576462> is a structure of type feature
   FEATURE-NAME:              sp1_face2
   FEATURE-FUNCTION:          insertor
   FEATURE-SHAPE:             rectangular-face
   FEATURE-DIMENSIONS:        (18.6 1.7)
   FEATURE-POSITION:          #2A((0 -5.1 0 1) (0 -1 0 0))
   FEATURE-BETA:              3.1415926535897936d0
   ADJACENT-FEATURES:         (sp1_face1)
```

```
    INCLUSIVE-FEATURES:          (sp1_h1 sp1_h2)

#<feature 1576565> is a structure of type feature
    FEATURE-NAME:                sp1_face3
    FEATURE-POSITION:            #2A((0 5.1 0 1) (0 1 0 0))
    ADJACENT-FEATURES:           (sp1_face1)
    INCLUSIVE-FEATURES:          (sp1_h3 sp1_h4)

#<feature 1576672> is a structure of type feature
    FEATURE-NAME:                sp1_h1
    FEATURE-FUNCTION:            container
    FEATURE-SHAPE:               cylindrical-hole
    FEATURE-DIMENSIONS:          (1.2 10.2)
    FEATURE-POSITION:            #2A((-5.6 -5.1 0 1) (0 -1 0 0))
    FEATURE-BETA:                0.0
    ENCOMPASSING-FEATURE:        sp1_face2

#<feature 1576777> is a structure of type feature
    FEATURE-NAME:                sp1_h2
    FEATURE-POSITION:            #2A((5.6 -5.1 0 1) (0 -1 0 0))
    ENCOMPASSING-FEATURE:        sp1_face2

#<feature 1577104> is a structure of type feature
    FEATURE-NAME:                sp1_h3
    FEATURE-POSITION:            #2A((-5.6 5.1 0 1) (0 1 0 0))
    ENCOMPASSING-FEATURE:        sp1_face3

#<feature 1577211> is a structure of type feature
    FEATURE-NAME:                sp1_h4
    FEATURE-POSITION:            #2A((5.6 5.1 0 1) (0 1 0 0))
    ENCOMPASSING-FEATURE:        sp1_face3
```

Information about small bolts:

```
#<object 11004144> is a structure of type object
    NAME:                        s_bolt1
    SHAPE:                       cylindrical
    LENGTH:                      1.3
    WIDTH:                       0.9
    HEIGHT:                      0.9
    ALPHA:                       #(0 0 1 6.2831853071795870)
    BETA:                        #(1 0 0 0)
    FEATURES:                    (sb1_thread sb1_shaft)
```

Modelled features of small bolts:

```
#<feature 11004251> is a structure of type feature
   FEATURE-NAME:                sb1_thread
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               thread
   FEATURE-DIMENSIONS:          (0.5 0.8 0.05 nil)
   FEATURE-POSITION:            #2A((-0.65 0 0 1) (-1 0 0 0))
   FEATURE-BETA:                0.0

#<feature 11004354> is a structure of type feature
   FEATURE-NAME:                sb1_shaft
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               cylindrical-shaft
   FEATURE-DIMENSIONS:          (0.5 0.8)
   FEATURE-POSITION:            #2A((-0.65 0 0 1) (-1 0 0 0))
   FEATURE-BETA:                0.0
```

Information about the top plate:

```
#<object 11000561> is a structure of type object
   NAME:                        top_plate
   SHAPE:                       rectangular-box
   LENGTH:                      18.6
   WIDTH:                       15.2
   HEIGHT:                      2.0
   ALPHA:                       #(0 0 1 6.283185307179587d0)
   BETA:                        #(0 1 0 6.283185307179587d0)
   FEATURES:                    (tp_bottom tp_h1 tp_h2
                                           tp_h3 tp_h4)
```

Input information pertaining to features of top_plate are given below. The features tp_h1, tp_h2, tp_h3, and tp_h4 are identical in all attributes except for their position. So, the input information pertaining to tp_h1 are presented in full, and only the varying position attribute is presented for the other three features.

```
#<feature 11000664> is a structure of type feature
   FEATURE-NAME:                tp_bottom
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               rectangular-face
   FEATURE-DIMENSIONS:          (18.6 15.2)
   FEATURE-POSITION:            #2A((0 0 -1.0 1) (0 0 -1 0))
   FEATURE-BETA:                3.14159265358979360
   INCLUSIVE-FEATURES:          (tp_h1 tp_h2 tp_h3 tp_h4)
```

```
#<feature 11000773> is a structure of type feature
   FEATURE-NAME:                tp_h1
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               cylindrical-hole
   FEATURE-DIMENSIONS:          (1.2 0.2)
   FEATURE-POSITION:            #2A((-5.5 -6.5 0.2 1) (0 0 1 0))
   FEATURE-BETA:                0.0
   ENCOMPASSING-FEATURE:        tp_bottom

#<feature 11001106> is a structure of type feature
   FEATURE-NAME:                tp_h2
   FEATURE-POSITION:            #2A((5.5 -6.5 0 1) (0 0 1 0))

#<feature 11001277> is a structure of type feature
   FEATURE-NAME:                tp_h3
   FEATURE-POSITION:            #2A((-5.5 6.5 0 1) (0 0 1 0))

#<feature 11001404> is a structure of type feature
   FEATURE-NAME:                tp_h4
   FEATURE-POSITION:            #2A((5.5 6.5 0 1) (0 0 1 0))
```

Information about large bolts:

```
#<object 11001525> is a structure of type object
   NAME:                        l_bolt1
   SHAPE:                       cylinder
   LENGTH:                      13.1
   WIDTH:                       1.4
   HEIGHT:                      1.4
   ALPHA:                       #(0 0 1 6.283185307179587d0)
   BETA:                        #(1 0 0 1.0471975511965979d0)
   FEATURES:                    (lb1_head lb1_shaft lb1_thread)
```

Modelled features of large bolts:

```
#<feature 11001631> is a structure of type feature
   FEATURE-NAME:                lb1_head
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               rectangular-face
   FEATURE-DIMENSIONS:          (1.4 1.4)
   FEATURE-POSITION:            #2A((6.25 0 0 1) (-1 0 0 0))
   FEATURE-BETA:                60
   ADJACENT-FEATURES:           (lb1_shaft)
```

```
#<feature 11001734> is a structure of type feature
   FEATURE-NAME:                lb1_shaft
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               cylindrical-shaft
   FEATURE-DIMENSIONS:          (1.2 9.7)
   FEATURE-POSITION:            #2A((-2.45 0 0 1) (-1 0 0 0))
   FEATURE-BETA:                0.0
   ADJACENT-FEATURES:           (lb1_head)

#<feature 11002041> is a structure of type feature
   FEATURE-NAME:                lb1_thread
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               thread
   FEATURE-DIMENSIONS:          (1.2 2.5 0.1 nil)
   FEATURE-POSITION:            #2A((-6.25 0 0 1) (-1 0 0 0))
   FEATURE-BETA:                0.0
```

E.2 A Partial Trace of ASSEMBLE's Processing of the Controller Assembly Task

In the course of building and modifying the goal network corresponding to this experimental assembly task, ASSEMBLE writes out intermediate states of the goal network. The complete trace produced by ASSEMBLE is rather large. However, when two goals $G1$ and $G2$ differ only in one of the objects they relate, and when the differing objects are themselves identical in all their attributes, the processing of goals $G1$ and $G2$ by the FIND-FEATURES subsystem are congruent. This is seen in the traces corresponding to earlier experiments (see Sections A.2, B.2 and C.2. The list of goals corresponding to the controller assembly task consists of several such groups of goals. The groupings are shown in Section E.1 detailing the input. The partial trace corresponding to controller assembly experiment presented below includes an account of the complete processing by the FIND-FEATURES subsystem of one of the goals from each of these groups. The hypothesized subgoals for the other goals are presented.

Program Input

```
--------------------------------------------------

List of Goals

1 : (thread l_bolt4 into base_plate)

2 : (thread l_bolt3 into base_plate)

3 : (thread l_bolt2 into base_plate)

4 : (thread l_bolt1 into base_plate)

5 : (insert l_bolt4 into side_plate_2)

                .
                .
                .

25 : (insert s_bolt1 into transformer)

26 : (place transformer on base_plate)


Ordering Constraints

Goal 26 precedes Goal 25.
Goal 25 precedes Goal 24.

          .
          .
          .

Goal 4 precedes Goal 3.
Goal 3 precedes Goal 2.
Goal 2 precedes Goal 1.

--------------------------------------------------


Creating feature level subgoals for object level goal

                (thread l_bolt4 into base_plate) ........
```

After applying the constraints associated with the operations the feature level subgoals are:

Feature Level Subgoals

(thread feature lb4_thread of l_bolt4 into feature lh4 of base_plate)

(thread feature lb4_thread of l_bolt4 into feature lh3 of base_plate)

(thread feature lb4_thread of l_bolt4 into feature lh2 of base_plate)

(thread feature lb4_thread of l_bolt4 into feature lh1 of base_plate)

Applying beta-symmetry of objects to reduce search tree ..

Applying alpha-symmetry of objects to reduce search tree..

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(thread feature lb4_thread of l_bolt4 into feature lh4 of base_plate)	0.25
(thread feature lb4_thread of l_bolt4 into feature lh3 of base_plate)	0.25
(thread feature lb4_thread of l_bolt4 into feature lh2 of base_plate)	0.25
(thread feature lb4_thread of l_bolt4 into feature lh1 of base_plate)	0.25

Functional symmetry:

Subgoal
(thread feature lb4_thread of l_bolt4 into feature lh3

```
                                          of side_plate_1)
(insert feature lb1_shaft of l_bolt1 into feature sp1_h1
                                          of side_plate_1)

(insert feature lb4_shaft of l_bolt4 into feature tp_h4
                                            of top_plate)
(insert feature lb3_shaft of l_bolt3 into feature tp_h3
                                            of top_plate)
(insert feature lb2_shaft of l_bolt2 into feature tp_h2
                                            of top_plate)
(insert feature lb1_shaft of l_bolt1 into feature tp_h1
                                            of top_plate)

(place feature tp_bottom of top_plate on feature sp2_face3
                                            of side_plate_2)
(place feature tp_bottom of top_plate on feature sp1_face3
                                            of side_plate_1)

(place feature sp2_face2 of side_plate_2 on feature plate_top
                                                 of base_plate)
(place feature sp1_face2 of side_plate_1 on feature plate_top
                                                 of base_plate)

(insert feature c_side of capacitor into feature hole
                                          of base_plate)

(thread feature sb4_thread of s_bolt4 into feature sh4
                                            of base_plate)
(thread feature sb3_thread of s_bolt3 into feature sh3
                                            of base_plate)
(thread feature sb2_thread of s_bolt2 into feature sh2
                                            of base_plate)
(thread feature sb1_thread of s_bolt1 into feature sh1
                                            of base_plate)

(insert feature sb4_shaft of s_bolt4 into feature h4
                                          of transformer)
(insert feature sb3_shaft of s_bolt3 into feature h2
                                          of transformer)
(insert feature sb2_shaft of s_bolt2 into feature h1
                                          of transformer)
(insert feature sb1_shaft of s_bolt1 into feature h3
                                          of transformer)

(place feature bottom of transformer on feature plate_top
```

```
of base_plate)
```

Subfeatures are generated corresponding to the mating features specified in each of the hypothesized subgoals listed above as per Table 5. Spatial relationships between these subfeatures are generated as per Table 6. The file of prolog assumptions containing declarative information about the subfeatures and spatial relationships that is generated by ASSEMBLE to be provided as input to RAPT includes 252 declarations. The file, however, was too large for ATMS, which acts as a front end for RAPT, to process. So, the assembly task was split up into two subassemblies:

> Subassembly-1, which consists of placing the transformer on the base_plate, fastening the base_plate and transformer with the four small bolts, and inserting the capacitor into the base_plate;
>
> Subassembly-2, which consists of placing the side_plates on the base_plate, placing the top_plate on top of the side_plates, and fastening the four objects with the four large bolts.

There are 114 assumptions corresponding to subassembly-1, and 141 assumptions corresponding to subassembly-2. They were both processed individually by RAPT. The individual input files and the results obtained are detailed in Appendix E.

7.5.1 Spatial Reasoning about Subassembly-1

In the case of subassembly-1, the alignments between the holes in the transformer and base_plate suggested by the groups of insert and thread operations involving the small bolts are spatially inconsistent. Figures 31 and 32 show the positions of the holes on the base_plate and transformer, respectively. Given this placement of the holes on the base_plate and transformer, there are two possible ways in which the holes can be matched to provide a spatially consistent alignment:

(sh1-h2), (sh2-h1), (sh3-h3), and (sh4-h4);

(sh1-h4), (sh2-h3), (sh3-h1), and (sh4-h2).

The hypothesized subgoals corresponding to the set of goals that accomplish the affixment operation are:

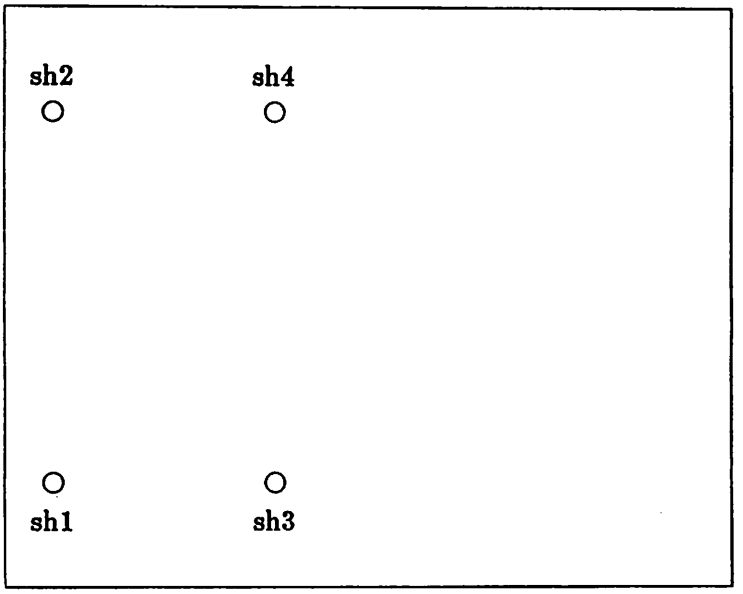


Figure 31: Positions of small holes on Base_Plate

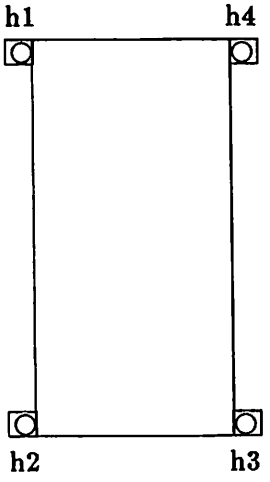


Figure 32: Positions of holes on Transformer

```
(thread feature sb4_thread of s_bolt4 into feature sh4
                                        of base_plate)
(thread feature sb3_thread of s_bolt3 into feature sh3
                                        of base_plate)
(thread feature sb2_thread of s_bolt2 into feature sh2
                                        of base_plate)
(thread feature sb1_thread of s_bolt1 into feature sh1
                                        of base_plate)

(insert feature sb4_shaft of s_bolt4 into feature h4
                                      of transformer)
(insert feature sb3_shaft of s_bolt3 into feature h2
                                      of transformer)
(insert feature sb2_shaft of s_bolt2 into feature h1
                                      of transformer)
(insert feature sb1_shaft of s_bolt1 into feature h3
                                      of transformer)
```

The alignments relationships between the holes on the two objects suggested by the set of hypothesized subgoals are:

(sh1-h3), (sh2-h1), (sh3-h2), (sh4-h4).

From Figures 31 and 32, it is evident that these alignment relationships are spatially inconsistent. These alignments partially coincide with the first of the two spatially consistent alignments listed above except that the alignments between sh1 and sh3, and h2 and h3 are mismatched.

As the output produced by RAPT corresponding to the subassembly (shown in Section E.3.1 of Appendix E) indicates, RAPT detects the inconsistency in the specified spatial relationships. Despite the inconsistency, aside from deducing the locations of the small bolts and capacitor, after several iterations, RAPT identifies the location of the transformer as well. The inferred location of the transformer is consistent with the alignments (sh2-h1) and (sh4-h4), and one that is acceptable for the successful completion of the assembly.

RAPT could infer the location of the transformer because the alignments (sh2-h1) and (sh4-h4) provide sufficient spatial constraints to determine this information. The erroneous alignments (sh1-h3) and (sh3-h2) are also mutually consistent, and provide

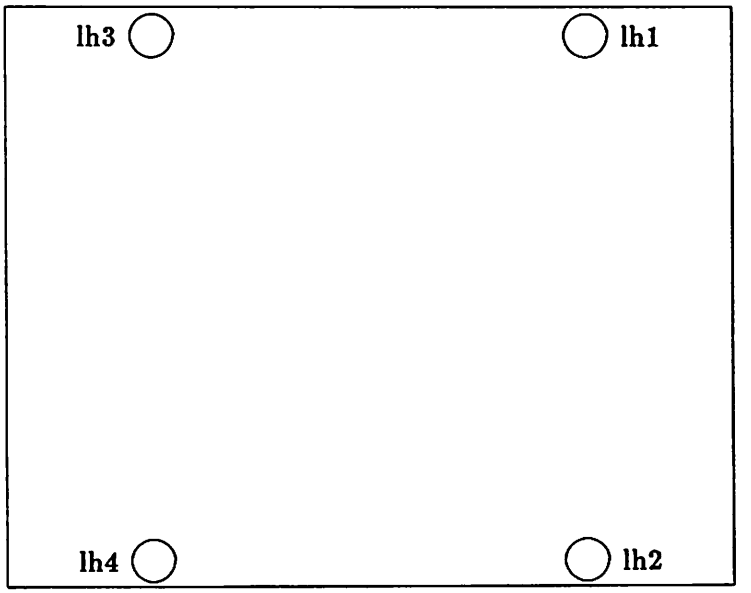


Figure 33: Positions of large holes on Base_Plate

sufficient spatial constraints to determine the pose of the transformer consistent with these alignments. In this case the resulting configuration has larger distance discrepancies. However, there is not sufficient evidence to support the theory that, given two sets of assumptions A and B such that the individual sets of assumptions are consistent, and $A \cup B$ is inconsistent, RAPT makes a choice between the consistent sets of assumptions based on a criterion by which some measure of the discrepancy is minimized.

7.5.2 Spatial Reasoning about Subassembly-2

In the case of subassembly-2, the alignment between the holes in the side_plates, top_plate, and base_plate are spatially inconsistent. Figures 33, 34, and 35 show the placement of the holes on the base_plate, top_plate and side_plate_1 which are to be aligned. The hypothesized subgoals suggest the following alignments:

(lh4 - sp2_h4 - tp_h4)

(lh3 - sp2_h1 - tp_h3)

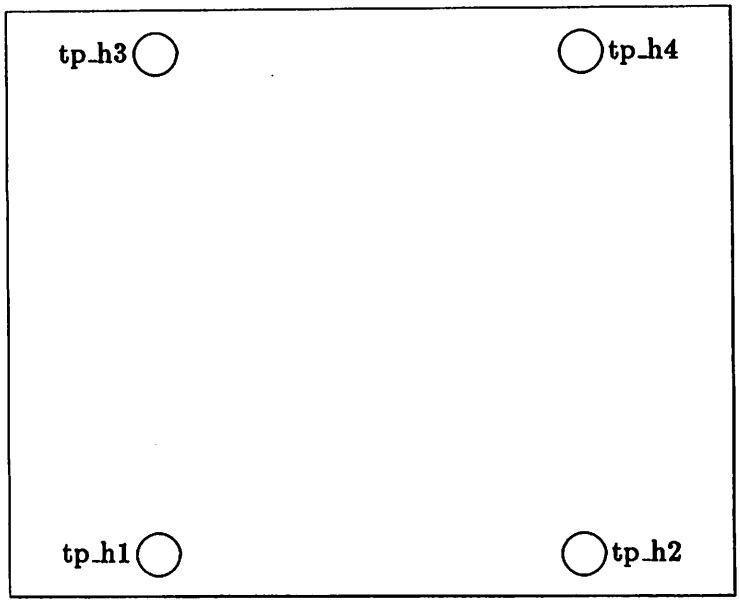


Figure 34: Positions of holes on Top_Plate

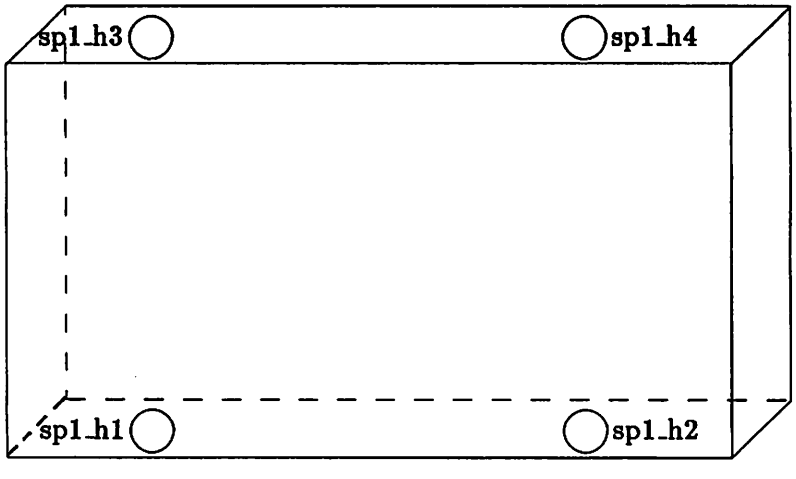


Figure 35: Positions of holes on Side_Plate_1

(lh2 - sp1_h4 - tp_h2)

(lh1 - sp1_h1 - tp_h1)

From Figures 33 and 34, the alignments (lh3-tp_h3) and (lh2-tp_h2) are not spatially consistent with alignments (lh1-tp_h1) and (lh4-tp_h4). Also, in aligning the holes on the top_plate and side_plate_1, tp_h1 and tp_h2 are matched with holes sp1_h1 and sp1_h4, which lie on opposite faces of the side_plate.

RAPT initially recognizes the inconsistencies in the spatial relationships between side_plate1 and top_plate, and subsequently the other inconsistencies as well. After several iterations, the system proposes a set of locations for the objects in the assembly. While the locations of the large bolts are identified correctly, the locations suggested by RAPT for the side_plates and top_plate are incorrect. Unlike the case of subassembly-1, there is not a sufficiently large subset of consistent assumptions in this case, to derive the location information correctly. Relevant portions of the output produced by RAPT corresponding to subassembly-2 are shown in Section E.5 of Appendix E.

7.6 Summary

This chapter has provided a detailed account of the experiments conducted to test the effectiveness of the object representations used and the reasoning processes employed. The implementations need to be extended to realize the proposed integration with motion planning systems. This section will attempt to summarize the effectiveness of the various procedures used in the FIND-FEATURES system to reduce the search space.

Table 7 summarizes the reduction in search space achieved by applying the geometric constraints associated with the operations, on each of the assembly experiments conducted. The number of subgoals is shown as a tuple (a, b), where appropriate, to indicate that a is the number of subgoals on the subgoals list and b is the number of subgoals on the choice-points list.

Table 8 presents the goal nodes corresponding to INSERT operations from the experimental assembly tasks on which the closeness-of-fit heuristic was effective in using the

Table 7: Search space reduction achieved by application of geometric constraints associated with operations and by application of object symmetry on the experimental assembly tasks

Operation	Size of cartesian-product set	Number of subgoals generated	Remaining subgoals after application of object symmetry
Experiment 1:			
Insert part_f in part_e	6	4	(1,1)
Insert part_g in part_e	6	4	(1,1)
Experiment 2:			
Insert part_i in part_h	3	2	1
Insert part_j in part_h	3	2	1
Experiment 3:			
Double-Insert part_l into part_k	21	5	5
Double-Insert part_m into part_k	21	16	8
Double-Insert part_n into part_k	21	16	8
Experiment 4:			
Insert part_b in part_a	16	3	3
Thread part_a with part_c	12	1	1
Thread part_b with part_d	16	2	2
Experiment 5:			
Place transformer on base_plate	60	2	2
Insert small-bolt in transformer	12	4	(2,2)
Thread small-bolt in base_plate	20	4	4
Insert capacitor into base_plate	20	2	2
Place side_plate on base_plate	70	3	(2,1)
Place top_plate on side_plate	35	3	(2,1)
Insert large-bolt into top_plate	15	4	4
Insert large-bolt into side_plate	15	4	(2,2)
Thread large-bolt into base_plate	15	4	4

Table 8: Goals for which the closeness-of-fit heuristic is effective in ranking the subgoals. The initial assignment of confidence measures and the modified confidence measures as a result of applying the heuristic are shown.

Goal node	subgoals	Initial confidence measure	Modified confidence measure
Experiment 3:			
Double-Insert part_m into part_k	((m3,k5) (m1,k6))	0.125	0.229
	((m3,k4) (m1,k5))	0.125	0.229
	((m3,k3) (m1,k4))	0.125	0.229
	((m3,k2) (m1,k7))	0.125	0.0625
	((m3,k2) (m1,k6))	0.125	0.0625
	((m3,k2) (m1,k5))	0.125	0.0625
	((m3,k2) (m1,k4))	0.125	0.0625
	((m3,k2) (m1,k3))	0.125	0.0625
Double-Insert part_n into part_k	((n3,k5) (n1,k6))	0.125	0.229
	((n3,k4) (n1,k5))	0.125	0.229
	((n3,k3) (n1,k4))	0.125	0.229
	((n3,k2) (n1,k7))	0.125	0.0625
	((n3,k2) (n1,k6))	0.125	0.0625
	((n3,k2) (n1,k5))	0.125	0.0625
	((n3,k2) (n1,k4))	0.125	0.0625
	((n3,k2) (n1,k3))	0.125	0.0625
Experiment 4:			
Insert part_b in part_a	(b4,a3)	0.3333	.16667
	(b2,a3)	0.3333	.16667
	(b1,a3)	0.3333	.66666

information to alter the confidence measures of subgoals. On experiments 1 and 2, the hypothesized subgoal is determined without application of heuristics, so they are not listed in the summary. In all other INSERT operations from among the experimental tasks, there is no variation in the dimensions of the mating features suggested by the various subgoals.

Table 9 summarizes the results of applying the functional symmetry heuristic on goal nodes on which the heuristic was effective in identifying equivalent subgoals and deferring such subgoals to the choice-points slot.

Table 9: Goals for which the functional symmetry heuristic was effective. The components of the tuples represent the number of subgoals in the subgoals slot and choice-points slot.

Operation	Remaining subgoals after application of object symmetry	Remaining subgoals after application of functional symmetry
Experiment 3:		
Double-Insert part_l into part_k	5	(1,4)
Double-Insert part_m into part_k	8	(2,6)
Double-Insert part_n into part_k	8	(2,6)
Experiment 4:		
Thread part_b with part_d	2	(1,1)
Experiment 5:		
Place transformer on base_plate	2	(1,1)
Insert small-bolt in transformer	(2,2)	(1,3)
Thread small-bolt into base_plate	4	(1,3)
Insert large-bolt into top_plate	4	(1,3)
Insert large-bolt into side_plate	(2,2)	(1,3)
Thread large-bolt into base_plate	4	(1,3)

The accessibility analysis has also proved effective in determining interactions between subgoals as is evident from the discussion of the experiments in earlier sections. This analysis needs to refined further in many respects as outlined in the discussion of the interaction analysis phase in Chapter 6. The usefulness of the spatial reasoning component is borne out by the two subassemblies that are a part of the controller assembly discussed in Section 7.5.

The experiments, specifically the two half-cylinders experiment as well as the light bulb fixture assembly experiment, have served to identify analysis of volumetric interference to be a necessary component of an assembly planning system. The development of an interface that will allow RAPT to communicate back pertinent information to ASSEMBLE and the integration of a solid modelling system into ASSEMBLE are seen as the near term extensions of the current version of ASSEMBLE.

CHAPTER 8

SUMMARY AND CONCLUSIONS

This chapter provides a summary of the work detailed in this dissertation, an outline of the specific contributions of the work, suggestions for possible extensions of the work, and some concluding remarks.

8.1 Summary of Work

The central problem this dissertation has addressed is one of representing and reasoning about objects abstractly. This is accomplished by defining the assembly problem as one where abstract and incomplete information about objects is provided and a system is required to reason with this information to arrive at a precise description of the assembly. Couching the general problem in the domain of assembly has allowed the exploitation of geometric constraints imposed by the assembly operations.

Studying the representation of and reasoning about objects in the context of the assembly planning problem has required an analysis of the basic underlying characteristics of assembly and the development of representations and related inferencing mechanisms based on these underlying characteristics. The characteristics of assembly that are emphasized in the representation of objects in this work are:

(a) the relevance of geometric information about the features of objects;

(b) the locality of such features on the object;

(c) object symmetry.

The representation of objects in terms of the volumes they represent has not received an equal emphasis, but is nevertheless a key aspect of reasoning about assembly. A hierarchical object model has been developed based on these characteristics.

Solving the assembly planning problem required:

(a) An effective modularization of the problem. This has been accomplished through the development of the hierarchical decomposition approach, where the assembly planning problem is restated in increasing levels of detail, first in terms of the features that will be related by the assembly, and further, in terms of the spatial relationships that will arise between the surfaces that constitute the features.

(b) Devising the reasoning mechanisms that will facilitate the transformations required by the hierarchical decomposition approach. As is to be expected, there is a tight coupling between representational capability allowed by the object models and the types of reasoning that is within reach of the assembly planning system. Feature geometry, object symmetry, geometric constraints arising from assembly operations, sequencing of the assembly operations, and locality of features are some of the information ingredients supporting the reasoning processes.

The extent of formality and determinism displayed by the various reasoning components of the system are different. Some of the components, such as the application of geometric constraints and spatial reasoning, and use of object symmetry are precise in determining the feasibility of a suggested assembly configuration, and determining the equivalence of suggested configurations. To an extent, this is due to the basic underlying philosophy adopted in this work, that using quick and somewhat imprecise mechanisms to generate reasonable hypotheses for solutions, and tuning the solutions using more formal mechanisms is an effective means of problem solving.

A secondary focus of this work has been the integration of the assembly planner developed in an intelligent robotic system. To this end, a system architecture was developed that interfaces the assembly planner with motion planning systems. Development of the system architecture has provided a context for the research on assembly planning. However, the architecture presented here has not been subjected to detailed analysis.

The development components of this work are primarily the realization of the object models and associated reasoning processes in an implemented system. The implemented system serves to substantiate that the models developed, the problem decomposition

adopted, and the reasoning strategies used are appropriate for the assembly planning problem.

8.2 Specific Contributions

The primary contributions of this work are:

- the definition of the assembly planning problem;
- the development of a system architecture suitable for robotics;
- the development of abstract object models for the purpose of planning assembly tasks and an evaluation of such models for reasoning about assembly;
- a hierarchical problem refinement approach to assembly planning that reasons about objects and assembly;
- the development of the program ASSEMBLE that realizes the object models developed and the hierarchical problem refinement approach to assembly planning.
- integration of the spatial reasoning system, RAPT, with ASSEMBLE.

8.3 Extensions and Future Work

A major limitation of the system identified in the discussion of the reasoning processes in Chapter 6 and further illustrated by the experiments discussed in Chapter 7 is the inability of the system to reason about the volumetric interference between objects. As suggested earlier, integrating a solid modeling system such as ROBMOD or PADL-2 into ASSEMBLE, in a manner similar to the integration of spatial reasoning into ASSEMBLE through the use of the RAPT system, can alleviate this problem. The object models have to be suitably extended. However, both the object models as well as the modular structure of the assembly planning system can easily accommodate such extensions. Integrating a solid modeler in the system will also help to tighten up some

of the less precise reasoning mechanisms of the system, such as the accessibility analysis phase and volumetric enclosure test.

In a similar vein, the symbolic descriptions of feature shapes can be augmented by precise mathematical descriptions of the surfaces. The accessibility analysis phase relies on the topological relationships defined between features of the same object. The reasoning is somewhat limited by the lack of information in the symbolic descriptions used. Use of more precise surface descriptions will provide a clearer interpretation of the feature relationships and this will, in turn, enhance the nature and extent of reasoning possible.

The extensions considered for the longer term relate to the integration of the assembly planning system with the motion planning systems as outlined in Chapter 3.

8.4 Concluding Remarks

The work described in this dissertation demonstrates that it is possible to reason about a task as precise and complex as assembling a set of parts, using geometric reasoning and some of the existing AI tools and techniques. The main stream of robotics research continues to address issues relating to the design and control of complex mechanisms, which is in the domain of engineering. The challenge robotics provides for researchers in Computer Science and Artificial Intelligence, however, is the development of the complex computational systems that will expand the capabilities of these mechanisms and invest some autonomy in these systems. This study has attempted to look at a problem that is very fundamental in this regard, namely, reasoning about objects in an effort to manipulate them. There is much work left to do, both within the context of the assembly planning problem discussed here, and in the larger context of building intelligent machines that can perceive and manipulate objects with ease.

APPENDIX A

THE PEGS IN HOLES EXPERIMENT

This appendix provides the input to and output from ASSEMBLE and the RAPT system for the simple Pegs in Holes assembly experiment described in Section 7.1. The appendix contains four sections: the input provided to ASSEMBLE, the trace ASSEMBLE produces in the course of executing the FIND-FEATURES subsystem and the accessibility analysis phase of the VERIFY subsystem, the file containing the prolog declarations that constitute the input to RAPT, and finally the output produced by RAPT.

A.1 Input to ASSEMBLE

The following constitutes the input to ASSEMBLE corresponding to the Pegs in Holes assembly experiment:

```
;;; load file containing object and feature information
(load "pegs.lisp")

;;; input assembly instructions
(insert part_f into part_e)
(insert part_g into part_e)

;;; precedence relations
(2 1)

;;; name of file to contain input to RAPT
"pegs.pl"

;;; object relative to which locations of other objects are deduced
part_e
```

A.1.1 Object and Feature Information

The structures generated for objects part_e, part_f, and part_g and their features from reading the input file pegs.lisp are shown below.

Input information pertaining to part_e, the circular plate:

```
#<object 1304656> is a structure of type object
   NAME:                        part_e
   SHAPE:                       circular-plate
   LENGTH:                      5.0
   WIDTH:                       5.0
   HEIGHT:                      1.0
   ALPHA:                       #(1 0 0 6.28318530717958 7d0)
   BETA:                        #(0 0 1 3.14159265358979 36d0)
   FEATURES:                    (e1 e2 e3)
```

The following structures correspond to the features of part_e:

```
#<feature 1305047> is a structure of type feature
   FEATURE-NAME:                e1
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               circular-face
   FEATURE-DIMENSIONS:          (5.0)
   FEATURE-POSITION:            #2A((0 0 0 1) (0 0 1 0))
   FEATURE-BETA:                180.0
   INCLUSIVE-FEATURES:          (e2 e3)

#<feature 1305074> is a structure of type feature
   FEATURE-NAME:                e2
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               cylindrical-hole
   FEATURE-DIMENSIONS:          (0.5 0.8)
   FEATURE-POSITION:            #2A((1.5 0.0 0.5 1.0) (0 0 1 0))
   FEATURE-BETA:                0.0
   ENCOMPASSING-FEATURE:        e1

#<feature 1305143> is a structure of type feature
   FEATURE-NAME:                e3
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               cylindrical-hole
   FEATURE-DIMENSIONS:          (0.5 0.8)
   FEATURE-POSITION:            #2A((-1.5 0.0 0.5 1.0) (0 0 1 0))
```

```
   FEATURE-BETA:                0.0
   ENCOMPASSING-FEATURE:        e1
```

Input information pertaining to part_f, one of the two identical cylindrical pegs:

```
#<object 1304741> is a structure of type object
   NAME:                        part_f
   SHAPE:                       cylinder
   LENGTH:                      4.0
   WIDTH:                       0.5
   HEIGHT:                      0.5
   ALPHA:                       #(0 1 0 3.141592653589793d0)
   BETA:                        #(1 0 0 0.0)
   FEATURES:                    (f1 f2)
```

Features of part_f:

```
#<feature 1305212> is a structure of type feature
   FEATURE-NAME:                f1
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               cylindrical-shaft
   FEATURE-DIMENSIONS:          (0.5 4.0)
   FEATURE-POSITION:            #2A((2.0 0.0 0.0 1.0) (1 0 0 0))
   FEATURE-BETA:                0.0
   OVERLAPPING-FEATURES:        (f2)

#<feature 1305263> is a structure of type feature
   FEATURE-NAME:                f2
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               cylindrical-shaft
   FEATURE-DIMENSIONS:          (0.5 4.0)
   FEATURE-POSITION:            #2A((-2.0 0.0 0.0 1.0) (-1 0 0 0))
   FEATURE-BETA:                0.0
   OVERLAPPING-FEATURES:        (f1)
```

The other cylindrical peg, part_g, is identical in all respects to part_f described above. So, although information about the object and its features is explicitly provided to the system, it is not repeated here.

A.2 A Trace of ASSEMBLE's Processing of the Task

In the course of building and modifying the goal network corresponding to this assembly task, ASSEMBLE writes out intermediate states of the goal network. The trace generated corresponding to the Pegs in Holes experiment is presented below:

```
--------------------------------------------------
                  Program Input
--------------------------------------------------

List of Goals

1 : (insert part_g into part_e)

2 : (insert part_f into part_e)


Ordering Constraints

Goal 2 precedes Goal 1.


--------------------------------------------------


Creating feature level subgoals for object level goal

                (insert part_g into part_e) ........


After applying the constraints associated with the operations
the feature level subgoals are:

      Feature Level Subgoals

      (insert feature g2 of part_g into feature e3 of part_e).

      (insert feature g2 of part_g into feature e2 of part_e).

      (insert feature g1 of part_g into feature e3 of part_e).

      (insert feature g1 of part_g into feature e2 of part_e).
```

Applying beta-symmetry of objects to reduce search tree ..

Feature Level Subgoals

(insert feature g2 of part_g into feature e3 of part_e).

(insert feature g1 of part_g into feature e3 of part_e).

Feature Level Subgoals deferred as Choice Points

(insert feature g2 of part_g into feature e2 of part_e).

(insert feature g1 of part_g into feature e2 of part_e).

Applying alpha-symmetry of objects to reduce search tree..

Feature Level Subgoals

(insert feature g2 of part_g into feature e3 of part_e).

Feature Level Subgoals deferred as Choice Points

(insert feature g2 of part_g into feature e2 of part_e).

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(insert feature g2 of part_g into feature e3 of part_e)	0.5

Deferred Feature Level Subgoals	Confidence Rates
(insert feature g2 of part_g into feature e2 of part_e)	0.5

Hypothesized subgoal:

(insert feature g2 of part_g into feature e3 of part_e).

Creating feature level subgoals for object level goal

(insert part_f into part_e) ....

After applying the constraints associated with the operations the feature level subgoals are:

Feature Level Subgoals

(insert feature f2 of part_f into feature e3 of part_e).

(insert feature f2 of part_f into feature e2 of part_e).

(insert feature f1 of part_f into feature e3 of part_e).

(insert feature f1 of part_f into feature e2 of part_e).

Applying beta-symmetry of objects to reduce search tree ..

Feature Level Subgoals

(insert feature f2 of part_f into feature e3 of part_e).

(insert feature f1 of part_f into feature e3 of part_e).

Feature Level Subgoals deferred as Choice Points

(insert feature f2 of part_f into feature e2 of part_e).

(insert feature f1 of part_f into feature e2 of part_e).

Applying alpha-symmetry of objects to reduce search tree..

Feature Level Subgoals

(insert feature f2 of part_f into feature e3 of part_e).

Feature Level Subgoals deferred as Choice Points

(insert feature f2 of part_f into feature e2 of part_e).

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(insert feature f2 of part_f into feature e3 of part_e)	0.5

Deferred Feature Level Subgoals	Confidence Rates
(insert feature f2 of part_f into feature e2 of part_e)	0.5

Hypothesized subgoal:

(insert feature f2 of part_f into feature e3 of part_e).

Accessibility Analysis

An alternative feature subgoal
(insert feature f2 of part_f into feature e2 of part_e)
is chosen for the object level goal
(insert part_f into part_e).

Tentative Plan at the Feature Level

(insert feature g2 of part_g into feature e3 of part_e).

```
(insert feature f2 of part_f into feature e2 of part_e).
```

A.3 RAPT Input Generated by ASSEMBLE

As specified in Section 6.6.2, ASSEMBLE generates subfeatures corresponding to the mating features, the requisite information for processing by RAPT including spatial relationships between subfeatures, and produces a file comprising a set of prolog assumptions. This file generated by ASSEMBLE for the Pegs in Holes experiment is presented below:

```
;;; initialization
:-init(assume).

;;; declarations pertaining to subfeatures of e3.

:-assume(e3_curved_face$part_e:cylindrical_face).
:-assume(r$e3_curved_face$part_e=0.25).
:-assume(loc$e3_curved_face$part_e=Pos_42),
       Pos_42 is stpos(0,0,1, 1,0,0, 0,1,0, -1.5,0.0,-0.3).
:-assume(e3_botttom_face$part_e:plane_face).
:-assume(loc$e3_botttom_face$part_e=Pos_43),
       Pos_43 is stpos(0,0,1, 1,0,0, 0,1,0, -1.5,0.0,-0.3).

;;; declarations pertaining to subfeatures of g2.

:-assume(g2_curved_face$part_g:cylindrical_face).
:-assume(r$g2_curved_face$part_g=0.25).
:-assume(loc$g2_curved_face$part_g=Pos_44),
       Pos_44 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.0,0.0,0.0).
:-assume(g2_botttom_face$part_g:plane_face).
:-assume(loc$g2_botttom_face$part_g=Pos_45),
       Pos_45 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.0,0.0,0.0).

;;; spatial relations between subfeatures of e3 and g2.

:-assume(g2_botttom_face$part_g against e3_botttom_face$part_e).
:-assume(g2_curved_face$part_g fits e3_curved_face$part_e).

;;; declarations pertaining to subfeatures of e2.
```

```
:-assume(e2_curved_face$part_e:cylindrical_face).
:-assume(r$e2_curved_face$part_e=0.25).
:-assume(loc$e2_curved_face$part_e=Pos_46),
       Pos_46 is stpos(0,0,1, 1,0,0, 0,1,0, 1.5,0.0,-0.3).
:-assume(e2_botttom_face$part_e:plane_face).
:-assume(loc$e2_botttom_face$part_e=Pos_47),
       Pos_47 is stpos(0,0,1, 1,0,0, 0,1,0, 1.5,0.0,-0.3).

;;; declarations pertaining to subfeatures of f2.

:-assume(f2_curved_face$part_f:cylindrical_face).
:-assume(r$f2_curved_face$part_f=0.25).
:-assume(loc$f2_curved_face$part_f=Pos_48),
       Pos_48 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.0,0.0,0.0).
:-assume(f2_botttom_face$part_f:plane_face).
:-assume(loc$f2_botttom_face$part_f=Pos_49),
       Pos_49 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.0,0.0,0.0).

;;; spatial relations between subfeatures of e2 and f2.

:-assume(f2_botttom_face$part_f against e2_botttom_face$part_e).
:-assume(f2_curved_face$part_f fits e2_curved_face$part_e).

;;; absolute location of part_e.

:-assume(loc$part_e=trans(0,0,0)).
```

A.4 Output Produced by RAPT

The output generated by RAPT, indicating the locations of part_f and part_g, based on the specified absolute location of part_e is given below:

```
The following relationships and locations will be used:
f2_botttom_face $ part_f against e2_botttom_face $ part_e
g2_botttom_face $ part_g against e3_botttom_face $ part_e
f2_curved_face $ part_f fits e2_curved_face $ part_e
g2_curved_face $ part_g fits e3_curved_face $ part_e
loc$part_e = pos(1,0,0; 0,1,0; 0,0,1; 0,0,0)

 Message from RAPT:
```

```
The following relationships and locations will be used:
f2_botttom_face $ part_f against e2_botttom_face $ part_e
g2_botttom_face $ part_g against e3_botttom_face $ part_e
f2_curved_face $ part_f fits e2_curved_face $ part_e
g2_curved_face $ part_g fits e3_curved_face $ part_e


The following locations have been inferred:


 Message from RAPT:


The following relationships and locations will be used:
f2_botttom_face $ part_f against e2_botttom_face $ part_e
g2_botttom_face $ part_g against e3_botttom_face $ part_e
f2_curved_face $ part_f fits e2_curved_face $ part_e
g2_curved_face $ part_g fits e3_curved_face $ part_e

 Message from RAPT:


The following relationships and locations will be used:
f2_botttom_face $ part_f against e2_botttom_face $ part_e
g2_botttom_face $ part_g against e3_botttom_face $ part_e
f2_curved_face $ part_f fits e2_curved_face $ part_e
g2_curved_face $ part_g fits e3_curved_face $ part_e
loc $ part_f = pos(0,0,1; 1,0,0; 0,1,0; 1.5,0.0,1.7)
loc $ part_e = pos(1,0,0; 0,1,0; 0,0,1; 0,0,0)
loc $ part_g = pos(0,0,1; 1,0,0; 0,1,0; -1.5,0.0,1.7)
```

The values of variables loc$part_f and loc$part_g give the locations of objects part_f and part_g, inferred by RAPT.

APPENDIX B

TWO HALF-CYLINDERS IN A CYLINDRICAL HOLE EXPERIMENT

This appendix provides the input to and output from the ASSEMBLE and RAPT systems for the Two Half-Cylinders in a Cylindrical Hole assembly experiment described in Section 7.2. The appendix contains four sections: the input provided to ASSEMBLE, the trace ASSEMBLE produces in the course of executing the FIND-FEATURES subsystem and the accessibility analysis phase of the VERIFY subsystem, the file containing the prolog declarations that constitute the input to RAPT, and finally the output produced by RAPT.

B.1 Input to ASSEMBLE

The following constitutes the input to ASSEMBLE corresponding to the Two Half-Cylinders in a Cylindrical Hole assembly experiment:

```
;;; load file containing object and feature information
(load "volume.lisp")

;;; input assembly instructions
(insert part_i into part_h)
(insert part_j into part_h)

;;; precedence relations
(2 1)

;;; name of file to contain input to RAPT
"volume.pl"

;;; object relative to which locations of other objects are deduced
part_h
```

B.1.1 Object and Feature Information

The structures generated for the objects part_h, part_i, and part_j and their features from reading the input file `volume.lisp` are shown below.
Input information pertaining to part_h, containing the cylindrical hole:

```
#<object 10300410> is a structure of type object
   NAME:                          part_h
   SHAPE:                         cylinder
   LENGTH:                        5.0
   WIDTH:                         3.0
   HEIGHT:                        3.0
   ALPHA:                         #(0 1 0 6.283185307179587d0)
   BETA:                          #(1 0 0 0.0)
   FEATURES:                      (h1)
```

The following structure corresponds to the single represented feature, h1, of part_h:

```
#<feature 10300515> is a structure of type feature
   FEATURE-NAME:                  h1
   FEATURE-FUNCTION:              container
   FEATURE-SHAPE:                 cylindrical-hole
   FEATURE-DIMENSIONS:            (2.8 4.9)
   FEATURE-POSITION:              #2A((2.5 0 0 1.0) (1 0 0 0))
   FEATURE-BETA:                  0.0
```

Input information pertaining to part_i, one of the two identical semi-cylindrical pegs:

```
#<object 10300633> is a structure of type object
   NAME:                          part_i
   SHAPE:                         semi-cylinder
   LENGTH:                        4.9
   WIDTH:                         2.8
   HEIGHT:                        1.4
   ALPHA:                         #(0 0 1 3.141592653589793 6d0)
   BETA:                          #(1 0 0 6.283185307179587d0)
   FEATURES:                      (i1 i2 i3)
```

Features of part_i:

```
#<feature 10300740> is a structure of type feature
   FEATURE-NAME:                i1
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               semi-circle
   FEATURE-DIMENSIONS:          (2.8)
   FEATURE-POSITION:            #2A((2.45 0 0.7 1.0) (1 0 0 0))
   FEATURE-BETA:                6.283185307179587d0
   ADJACENT-FEATURES:           (i2)

#<feature 10301025> is a structure of type feature
   FEATURE-NAME:                i2
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               semi-cyl-surface
   FEATURE-DIMENSIONS:          (2.8 4.9)
   FEATURE-POSITION:            #2A((0 0 0 1) (0 0 1 0))
   FEATURE-BETA:                6.283185307179587d0
   ADJACENT-FEATURES:           (i1 i3)

#<feature 10301221> is a structure of type feature
   FEATURE-NAME:                i3
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               semi-circle
   FEATURE-DIMENSIONS:          (2.8)
   FEATURE-POSITION:            #2A((-2.45 0 0.7 1.0) (-1 0 0 0))
   FEATURE-BETA:                6.2831853071795870
   ADJACENT-FEATURES:           (i2)
```

The other semi-cylindrical peg, part_j, is identical in all respects to part_i described above. So, although information about the object and its features is explicitly provided to the system, it is not repeated here.

B.2 A Trace of ASSEMBLE's Processing of the Task

In the course of building and modifying the goal network corresponding to this experimental assembly task, ASSEMBLE writes out intermediate states of the goal network. The trace generated corresponding to the Two Half-Cylinders in a Cylindrical Hole experiment is presented below:

```
                    Program Input
-----------------------------------------------------

List of Goals

1 : (insert part_j into part_h)

2 : (insert part_i into part_h)


Ordering Constraints

Goal 2 precedes Goal 1.

-----------------------------------------------------


Creating feature level subgoals for object level goal

                (insert part_j into part_h) ........


After applying the constraints associated with the operations
the feature level subgoals are:

      Feature Level Subgoals

      (insert feature j3 of part_j into feature h1 of part_h).

      (insert feature j1 of part_j into feature h1 of part_h).


Applying beta-symmetry of objects to reduce search tree ..


Applying alpha-symmetry of objects to reduce search tree..

      Feature Level Subgoals

      (insert feature j3 of part_j into feature h1 of part_h).
```

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(insert feature j3 of part_j into feature h1 of part_h)	1.0

Creating feature level subgoals for object level goal

(insert part_i into part_h) .....

After applying the constraints associated with the operations
the feature level subgoals are:

Feature Level Subgoals

(insert feature i3 of part_i into feature h1 of part_h).

(insert feature i1 of part_i into feature h1 of part_h).

Applying beta-symmetry of objects to reduce search tree ..

Applying alpha-symmetry of objects to reduce search tree..

Feature Level Subgoals

(insert feature i3 of part_i into feature h1 of part_h).

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates

```
(insert feature i3 of part_i into feature h1 of part_h)     1.0

-----------------------------------------------------
                Accessibility Analysis
-----------------------------------------------------


-----------------------------------------------------
          Tentative Plan at the Feature Level
-----------------------------------------------------

       (insert feature j3 of part_j into feature h1 of part_h).

       (insert feature i3 of part_i into feature h1 of part_h).
```

B.3 RAPT Input Generated by ASSEMBLE

As specified in Section 6.6.2, ASSEMBLE generates subfeatures corresponding to the mating features, the requisite information for processing by RAPT including spatial relationships between subfeatures, and produces a file comprising a set of prolog assumptions. This file generated by ASSEMBLE for the Two Half-Cylinders in a Cylindrical Hole experiment is presented below:

```
;;; initialization
:-init(assume).

;;; declarations pertaining to subfeatures of h1.

:-assume(h1_curved_face$part_h:cylindrical_face).
:-assume(r$h1_curved_face$part_h=1.4).
:-assume(loc$h1_curved_face$part_h=Pos_50),
        Pos_50 is stpos(1,0,0, 0,1,0, 0,0,1, -2.4,0.0,0.0).
:-assume(h1_botttom_face$part_h:plane_face).
:-assume(loc$h1_botttom_face$part_h=Pos_51),
        Pos_51 is stpos(1,0,0, 0,1,0, 0,0,1, -2.4,0.0,0.0).

;;; declarations pertaining to subfeatures of j3.
```

```
:-assume(j3_curved_face$part_j:cylindrical_face).
:-assume(r$j3_curved_face$part_j=1.4).
:-assume(loc$j3_curved_face$part_j=Pos_52),
       Pos_52 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0.7).
:-assume(j3_botttom_face$part_j:plane_face).
:-assume(loc$j3_botttom_face$part_j=Pos_53),
       Pos_53 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0.7).

;;; spatial relations between subfeatures of h1 and j3.

:-assume(j3_botttom_face$part_j against h1_botttom_face$part_h).
:-assume(j3_curved_face$part_j fits h1_curved_face$part_h).

;;; declarations pertaining to subfeatures of i3.

:-assume(i3_curved_face$part_i:cylindrical_face).
:-assume(r$i3_curved_face$part_i=1.4).
:-assume(loc$i3_curved_face$part_i=Pos_56),
       Pos_56 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0.7).
:-assume(i3_botttom_face$part_i:plane_face).
:-assume(loc$i3_botttom_face$part_i=Pos_57),
       Pos_57 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0.7).

;;; spatial relations between subfeatures of h1 and i3.

:-assume(i3_botttom_face$part_i against h1_botttom_face$part_h).
:-assume(i3_curved_face$part_i fits h1_curved_face$part_h).

;;; absolute location of part_h.

:-assume(loc$part_h=trans(0,0,0)).
```

B.4 Output Produced by RAPT

The output generated by RAPT, indicating the locations of part_i and part_j, based on the location specification of part_h is given below:

```
The following relationships and locations will be used:
i3_botttom_face $ part_i against h1_botttom_face $ part_h
j3_botttom_face $ part_j against h1_botttom_face $ part_h
```

```
i3_curved_face $ part_i fits h1_curved_face $ part_h
j3_curved_face $ part_j fits h1_curved_face $ part_h
loc$part_h = pos(1,0,0; 0,1,0; 0,0,1; 0,0,0)

 Message from RAPT:


The following relationships and locations will be used:
i3_botttom_face $ part_i against h1_botttom_face $ part_h
j3_botttom_face $ part_j against h1_botttom_face $ part_h
i3_curved_face $ part_i fits h1_curved_face $ part_h
j3_curved_face $ part_j fits h1_curved_face $ part_h


The following locations have been inferred:


 Message from RAPT:


The following relationships and locations will be used:
i3_botttom_face $ part_i against h1_botttom_face $ part_h
j3_botttom_face $ part_j against h1_botttom_face $ part_h
i3_curved_face $ part_i fits h1_curved_face $ part_h
j3_curved_face $ part_j fits h1_curved_face $ part_h

 Message from RAPT:


The following relationships and locations will be used:
i3_botttom_face $ part_i against h1_botttom_face $ part_h
j3_botttom_face $ part_j against h1_botttom_face $ part_h
i3_curved_face $ part_i fits h1_curved_face $ part_h
j3_curved_face $ part_j fits h1_curved_face $ part_h
loc $ part_i = pos(1,0,0; 0,1,0; 0,0,1; 0.05,0.0,-0.7)
loc $ part_h = pos(1,0,0; 0,1,0; 0,0,1; 0,0,0)
loc $ part_j = pos(1,0,0; 0,1,0; 0,0,1; 0.05,0.0,-0.7)
```

The values of variables loc$part_i and loc$part_j give the locations of objects part_i and part_j, inferred by RAPT.

APPENDIX C

DOUBLE-INSERTIONS EXPERIMENT

This appendix provides the input to and output from ASSEMBLE and the RAPT system for the Double-Insertions assembly experiment described in Section 7.3. The appendix contains four sections: the input provided to ASSEMBLE, the trace ASSEMBLE produces in the course of executing the FIND-FEATURES subsystem and the accessibility analysis phase of the VERIFY subsystem, the file containing the prolog declarations that constitute the input to RAPT, and finally the output produced by RAPT.

C.1 Input to ASSEMBLE

The following constitutes the input to ASSEMBLE corresponding to the Double-Insertions assembly experiment:

```
;;; load file containing object and feature information
(load "doubleins.lisp")

;;; input assembly instructions
(double-insert part_l into part_k)
(double-insert part_m into part_k)
(double-insert part_n into part_k)

;;; precedence relations
(2 1) (3 2)

;;; name of file to contain input to RAPT
"doubleins.pl"

;;; object relative to which locations of other objects are deduced
part_k
```

C.1.1 Object and Feature Information

The structures generated for the objects part_k, part_l, and part_m and their features from reading the input file `doubleins.lisp` are shown below.

Input information pertaining to part_k, the rectangular plate:

```
#<object 4746067> is a structure of type object
   NAME:                          part_k
   SHAPE:                         rectangular-plate
   LENGTH:                        6.0
   WIDTH:                         6.0
   HEIGHT:                        1.0
   ALPHA:                         #(1 0 0 6.283185307179587d0)
   BETA:                          #(0 0 1 6.283185307179587d0)
   FEATURES:                      (k1 k2 k3 k4 k5 k6 k7)
```

The following structures correspond to the features of part_k:

```
#<feature 4746171> is a structure of type feature
   FEATURE-NAME:                  k1
   FEATURE-FUNCTION:              insertor
   FEATURE-SHAPE:                 rectangular-face
   FEATURE-DIMENSIONS:            (6.0 6.0)
   FEATURE-POSITION:              #2A((0 0 1 1) (0 0 1 0))
   FEATURE-BETA:                  90
   INCLUSIVE-FEATURES:            (k2 k3 k4 k5 k6 k7)

#<feature 4746300> is a structure of type feature
   FEATURE-NAME:                  k2
   FEATURE-FUNCTION:              container
   FEATURE-SHAPE:                 cylindrical-hole
   FEATURE-DIMENSIONS:            (2.0 1.0)
   FEATURE-POSITION:              #2A((0 0 1 1) (0 0 1 0))
   FEATURE-BETA:                  0.0
   ENCOMPASSING-FEATURE:          k1

#<feature 4746463> is a structure of type feature
   FEATURE-NAME:                  k3
   FEATURE-FUNCTION:              container
   FEATURE-SHAPE:                 cylindrical-hole
   FEATURE-DIMENSIONS:            (0.5 1.0)
```

```
   FEATURE-POSITION:            #2A((-2.4142137 -1.0 1 1)
                                    (0 0 1 0))
   FEATURE-BETA:                0.0
   ENCOMPASSING-FEATURE:        k1

#<feature 4746566> is a structure of type feature
   FEATURE-NAME:                k4
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               cylindrical-hole
   FEATURE-DIMENSIONS:          (0.5 1.0)
   FEATURE-POSITION:            #2A((-2.4142137 1.0 1 1)
                                    (0 0 1 0))
   FEATURE-BETA:                0.0
   ENCOMPASSING-FEATURE:        k1

#<feature 4746671> is a structure of type feature
   FEATURE-NAME:                k5
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               cylindrical-hole
   FEATURE-DIMENSIONS:          (0.5 1.0)
   FEATURE-POSITION:            #2A((-1.0 2.4142137 1 1)
                                    (0 0 1 0))
   FEATURE-BETA:                0.0
   ENCOMPASSING-FEATURE:        k1

#<feature 4746772> is a structure of type feature
   FEATURE-NAME:                k6
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               cylindrical-hole
   FEATURE-DIMENSIONS:          (0.5 1.0)
   FEATURE-POSITION:            #2A((1.0 2.4142137 1 1)
                                    (0 0 1 0))
   FEATURE-BETA:                0.0
   ENCOMPASSING-FEATURE:        k1

#<feature 5011052> is a structure of type feature
   FEATURE-NAME:                k7
   FEATURE-FUNCTION:            container
   FEATURE-SHAPE:               cylindrical-hole
   FEATURE-DIMENSIONS:          (0.5 1.0)
   FEATURE-POSITION:            #2A((1.847759 -1.847759 1 1)
                                    (0 0 1 0))
   FEATURE-BETA:                0.0
   ENCOMPASSING-FEATURE:        k1
```

Input information pertaining to part_1, containing two asymmetrically sized shafts:

```
#<object 4747215> is a structure of type object
   NAME:                        part_1
   SHAPE:                       nil
   LENGTH:                      3.0
   WIDTH:                       2.0
   HEIGHT:                      2.0
   ALPHA:                       #(0 1 0 6.283185307179587d0)
   BETA:                        #(0 0 1 6.283185307179587d0)
   FEATURES:                    (l1 l2 l3)
```

Features of part_1:

```
#<feature 4747322> is a structure of type feature
   FEATURE-NAME:                l1
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               cylindrical-shaft
   FEATURE-DIMENSIONS:          (2.0 2.0)
   FEATURE-POSITION:            #2A((-1.6815629 0 -2.25 1) (0 0 -1 0))
   FEATURE-BETA:                0.0
   ADJACENT-FEATURES:           (l2)

#<feature 4747471> is a structure of type feature
   FEATURE-NAME:                l2
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               cylindrical-surface
   FEATURE-DIMENSIONS:          (0.5 1.3631258)
   FEATURE-POSITION:            #2A((0 0 0 1) (0 0 1 0))
   FEATURE-BETA:                0.0
   ADJACENT-FEATURES:           (l1 l3)

#<feature 4747576> is a structure of type feature
   FEATURE-NAME:                l3
   FEATURE-FUNCTION:            insertor
   FEATURE-SHAPE:               cylindrical-shaft
   FEATURE-DIMENSIONS:          (0.5 2.0)
   FEATURE-POSITION:            #2A((0.9315629 0 -2.25 1)
                                    (0 0 -1 0))
   FEATURE-BETA:                0.0
   ADJACENT-FEATURES:           (l2)
```

Input information pertaining to part_m, one of the two identical objects having shafts of identical size:

```
#<object 4747713> is a structure of type object
   NAME:                          part_m
   SHAPE:                         nil
   LENGTH:                        2.5
   WIDTH:                         1.5
   HEIGHT:                        0.5
   ALPHA:                         #(0 1 0 6.283185307179587d0)
   BETA:                          #(1 0 0 3.141592653589793d0)
   FEATURES:                      (m1 m2 m3)
```

Features of part_m:

```
#<feature 4750020> is a structure of type feature
   FEATURE-NAME:                  m1
   FEATURE-FUNCTION:              insertor
   FEATURE-SHAPE:                 cylindrical-shaft
   FEATURE-DIMENSIONS:            (0.5 2.0)
   FEATURE-POSITION:              #2A((-2.25 -1.0 0 1) (-1 0 0 0))
   FEATURE-BETA:                  0.0
   ADJACENT-FEATURES:             (m2)

#<feature 4750103> is a structure of type feature
   FEATURE-NAME:                  m2
   FEATURE-FUNCTION:              insertor
   FEATURE-SHAPE:                 cylindrical-surface
   FEATURE-DIMENSIONS:            (0.5 1.5)
   FEATURE-POSITION:              #2A((0 0 0 1) (0 0 1 0))
   FEATURE-BETA:                  0.0
   ADJACENT-FEATURES:             (m1 m3)

#<feature 4750210> is a structure of type feature
   FEATURE-NAME:                  m3
   FEATURE-FUNCTION:              insertor
   FEATURE-SHAPE:                 cylindrical-shaft
   FEATURE-DIMENSIONS:            (0.5 2.0)
   FEATURE-POSITION:              #2A((-2.25 1.0 0 1) (-1 0 0 0))
   FEATURE-BETA:                  0.0
   ADJACENT-FEATURES:             (m2)
```

The other symmetrical object, part_n, is identical in all respects to part_m described above. So, although information about the object and its features is explicitly provided to the system, it is not repeated here.

C.2 A Trace of ASSEMBLE's Processing of the Task

In the course of building and modifying the goal network corresponding to this experimental assembly task, ASSEMBLE writes out intermediate states of the goal network. The trace generated corresponding to the Double-Insertions experiment is presented below:

```
-----------------------------------------------------
                    Program Input
-----------------------------------------------------

 List of Goals

 1 : (double-insert part_n into part_k)

 2 : (double-insert part_m into part_k)

 3 : (double-insert part_l into part_k)


 Ordering Constraints

 Goal 2 precedes Goal 1.
 Goal 3 precedes Goal 2.


-----------------------------------------------------


 Creating feature level subgoals for object level goal

                 (double-insert part_n into part_k) ........


 After applying the constraints associated with the operations
 the feature level subgoals are:

Feature Level Subgoals

(double-insert feature n3 of part_n into feature k5 of part_k
                              and feature n1 into feature k6).
```

```
(double-insert feature n1 of part_n into feature k5 of part_k
                              and feature n3 into feature k6).

(double-insert feature n3 of part_n into feature k4 of part_k
                              and feature n1 into feature k5).

(double-insert feature n1 of part_n into feature k4 of part_k
                              and feature n3 into feature k5).

(double-insert feature n3 of part_n into feature k3 of part_k
                              and feature n1 into feature k4).

(double-insert feature n1 of part_n into feature k3 of part_k
                              and feature n3 into feature k4).

(double-insert feature n3 of part_n into feature k2 of part_k
                              and feature n1 into feature k7).

(double-insert feature n1 of part_n into feature k2 of part_k
                              and feature n3 into feature k7).

(double-insert feature n3 of part_n into feature k2 of part_k
                              and feature n1 into feature k6).

(double-insert feature n1 of part_n into feature k2 of part_k
                              and feature n3 into feature k6).

(double-insert feature n3 of part_n into feature k2 of part_k
                              and feature n1 into feature k5).

(double-insert feature n1 of part_n into feature k2 of part_k
                              and feature n3 into feature k5).

(double-insert feature n3 of part_n into feature k2 of part_k
                              and feature n1 into feature k4).

(double-insert feature n1 of part_n into feature k2 of part_k
                              and feature n3 into feature k4).

(double-insert feature n3 of part_n into feature k2 of part_k
                              and feature n1 into feature k3).

(double-insert feature n1 of part_n into feature k2 of part_k
                              and feature n3 into feature k3).
```

```
Applying beta-symmetry of objects to reduce search tree ..

Feature Level Subgoals

(double-insert feature n3 of part_n into feature k5 of part_k
                              and feature n1 into feature k6).

(double-insert feature n3 of part_n into feature k4 of part_k
                              and feature n1 into feature k5).

(double-insert feature n3 of part_n into feature k3 of part_k
                              and feature n1 into feature k4).

(double-insert feature n3 of part_n into feature k2 of part_k
                              and feature n1 into feature k7).

(double-insert feature n3 of part_n into feature k2 of part_k
                              and feature n1 into feature k6).

(double-insert feature n3 of part_n into feature k2 of part_k
                              and feature n1 into feature k5).

(double-insert feature n3 of part_n into feature k2 of part_k
                              and feature n1 into feature k4).

(double-insert feature n3 of part_n into feature k2 of part_k
                              and feature n1 into feature k3).


Applying alpha-symmetry of objects to reduce search tree..


 Subgoals with intial confidence rate assignments:

 Feature Level Subgoals                                    Confidence
                                                             Rates

(double-insert feature n3 of part_n into feature k5 of part_k
          and feature n1 into feature k6)                    0.125

(double-insert feature n3 of part_n into feature k4 of part_k
          and feature n1 into feature k5)                    0.125

(double-insert feature n3 of part_n into feature k3 of part_k
```

```
                                          of base_plate)
is moved to the list of choice points. There is a functionally
symmetrical alternative in
(thread feature lb4_thread of l_bolt4 into feature lh4
                                          of base_plate).

Subgoal
(thread feature lb4_thread of l_bolt4 into feature lh2
                                          of base_plate)
is moved to the list of choice points. There is a functionally
symmetrical alternative in
(thread feature lb4_thread of l_bolt4 into feature lh4
                                          of base_plate).

Subgoal
(thread feature lb4_thread of l_bolt4 into feature lh1
                                          of base_plate)
is moved to the list of choice points. There is a functionally
symmetrical alternative in
(thread feature lb4_thread of l_bolt4 into feature lh4
                                          of base_plate).

Hypothesized subgoal:

(thread feature lb4_thread of l_bolt4 into feature lh4
                                          of base_plate).

--------------------------------------------------


 Creating feature level subgoals for object level goal

                (thread l_bolt3 into base_plate) ........

                                .
                                .
                                .
                                .
                                .
Hypothesized subgoal:

(thread feature lb3_thread of l_bolt3 into feature lh4
                                          of base_plate).

--------------------------------------------------
```

```
 Creating feature level subgoals for object level goal

                (thread l_bolt2 into base_plate) ........

                          .
                          .
                          .
                          .
                          .

Hypothesized subgoal:

(thread feature lb2_thread of l_bolt2 into feature lh4
                                        of base_plate).

-----------------------------------------------------

 Creating feature level subgoals for object level goal

                (thread l_bolt1 into base_plate) ........

                          .
                          .
                          .
                          .
                          .

Hypothesized subgoal:

(thread feature lb1_thread of l_bolt1 into feature lh4
                                        of base_plate).

-----------------------------------------------------

Creating feature level subgoals for object level goal

                (insert l_bolt4 into side_plate_2) ........


 After applying the constraints associated with the operations
```

the feature level subgoals are:

Feature Level Subgoals

(insert feature lb4_shaft of l_bolt4 into feature sp2_h4
of side_plate_2).

(insert feature lb4_shaft of l_bolt4 into feature sp2_h3
of side_plate_2).

(insert feature lb4_shaft of l_bolt4 into feature sp2_h2
of side_plate_2).

(insert feature lb4_shaft of l_bolt4 into feature sp2_h1
of side_plate_2).

Applying beta-symmetry of objects to reduce search tree ..

Applying alpha-symmetry of objects to reduce search tree..

Feature Level Subgoals

(insert feature lb4_shaft of l_bolt4 into feature sp2_h4
of side_plate_2).

(insert feature lb4_shaft of l_bolt4 into feature sp2_h3
of side_plate_2).

Feature Level Subgoals deferred as Choice Points

(insert feature lb4_shaft of l_bolt4 into feature sp2_h1
of side_plate_2).

(insert feature lb4_shaft of l_bolt4 into feature sp2_h2
of side_plate_2).

Subgoals with intial confidence rate assignments:

Feature Level Subgoals Confidence Rates

```
(insert feature lb4_shaft of l_bolt4 into
        feature sp2_h4 of side_plate_2)                    0.25

(insert feature lb4_shaft of l_bolt4 into
        feature sp2_h3 of side_plate_2)                    0.25

Deferred Feature Level Subgoals                          Confidence
                                                            Rates

(insert feature lb4_shaft of l_bolt4 into
        feature sp2_h1 of side_plate_2)                    0.25

(insert feature lb4_shaft of l_bolt4 into
        feature sp2_h2 of side_plate_2)                    0.25

Functional symmetry:

Subgoal
(insert feature lb4_shaft of l_bolt4 into feature sp2_h3
                                        of side_plate_2)
is moved to the list of choice points. There is a functionally
symmetrical alternative in
(insert feature lb4_shaft of l_bolt4 into feature sp2_h4
                                        of side_plate_2).

Hypothesized subgoal:

(insert feature lb4_shaft of l_bolt4 into feature sp2_h4
                                        of side_plate_2).

-----------------------------------------------------


 Creating feature level subgoals for object level goal

                (insert l_bolt3 into side_plate_2) ........

                        .
                        .
                        .
                        .
                        .

Hypothesized subgoal:
```

```
(insert feature lb3_shaft of l_bolt3 into feature sp2_h4
                                          of side_plate_2).

-----------------------------------------------------

 Creating feature level subgoals for object level goal

                  (insert l_bolt2 into side_plate_1) ........

                                  .
                                  .
                                  .
                                  .
                                  .

Hypothesized subgoal:

(insert feature lb2_shaft of l_bolt2 into feature sp1_h4
                                        of side_plate_1).


-----------------------------------------------------

 Creating feature level subgoals for object level goal

                  (insert l_bolt1 into side_plate_1) ........

                                  .
                                  .
                                  .
                                  .
                                  .

Hypothesized subgoal:

(insert feature lb1_shaft of l_bolt1 into feature sp1_h4
                                        of side_plate_1).

-----------------------------------------------------
```

Creating feature level subgoals for object level goal

(insert l_bolt4 into top_plate) .....

After applying the constraints associated with the operations the feature level subgoals are:

Feature Level Subgoals

(insert feature lb4_shaft of l_bolt4 into feature tp_h4 of top_plate).

(insert feature lb4_shaft of l_bolt4 into feature tp_h3 of top_plate).

(insert feature lb4_shaft of l_bolt4 into feature tp_h2 of top_plate).

(insert feature lb4_shaft of l_bolt4 into feature tp_h1 of top_plate).

Applying beta-symmetry of objects to reduce search tree ..

Applying alpha-symmetry of objects to reduce search tree..

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(insert feature lb4_shaft of l_bolt4 into feature tp_h4 of top_plate)	0.25
(insert feature lb4_shaft of l_bolt4 into feature tp_h3 of top_plate)	0.25
(insert feature lb4_shaft of l_bolt4 into feature tp_h2 of top_plate)	0.25
(insert feature lb4_shaft of l_bolt4 into	

```
        feature tp_h1 of top_plate)                          0.25

Functional symmetry:

Subgoal
(insert feature lb4_shaft of l_bolt4 into feature tp_h3
                                          of top_plate)
is moved to the list of choice points. There is a functionally
symmetrical alternative in
(insert feature lb4_shaft of l_bolt4 into feature tp_h4
                                          of top_plate).

Subgoal
(insert feature lb4_shaft of l_bolt4 into feature tp_h2
                                          of top_plate)
is moved to the list of choice points. There is a functionally
symmetrical alternative in
(insert feature lb4_shaft of l_bolt4 into feature tp_h4
                                          of top_plate).

Subgoal
(insert feature lb4_shaft of l_bolt4 into feature tp_h1
                                          of top_plate)
is moved to the list of choice points. There is a functionally
symmetrical alternative in
(insert feature lb4_shaft of l_bolt4 into feature tp_h4
                                          of top_plate).

Hypothesized subgoal:

(insert feature lb4_shaft of l_bolt4 into feature tp_h4
                                          of top_plate).

-------------------------------------------------

 Creating feature level subgoals for object level goal

                (insert l_bolt3 into top_plate) ........
```

```
Hypothesized subgoal:

(insert feature lb3_shaft of l_bolt3 into feature tp_h4
                                        of top_plate).

-------------------------------------------------

 Creating feature level subgoals for object level goal

                (insert l_bolt2 into top_plate) ........

                        .
                        .
                        .
                        .
                        .

Hypothesized subgoal:

(insert feature lb2_shaft of l_bolt2 into feature tp_h4
                                        of top_plate).

-------------------------------------------------

 Creating feature level subgoals for object level goal

                (insert l_bolt1 into top_plate) ........

                        .
                        .
                        .
                        .
                        .

Hypothesized subgoal:

(insert feature lb1_shaft of l_bolt1 into feature tp_h4
                                        of top_plate).

-------------------------------------------------
```

```
Creating feature level subgoals for object level goal

                (place top_plate on side_plate_2) ........


After applying the constraints associated with the operations
the feature level subgoals are:

Feature Level Subgoals

(place feature tp_bottom of top_plate on feature sp2_face3
                                          of side_plate_2).

(place feature tp_bottom of top_plate on feature sp2_face2
                                          of side_plate_2).

(place feature tp_bottom of top_plate on feature sp2_face1
                                          of side_plate_2).


Applying beta-symmetry of objects to reduce search tree ..


Applying alpha-symmetry of objects to reduce search tree..

Feature Level Subgoals

(place feature tp_bottom of top_plate on feature sp2_face3
                                          of side_plate_2).

(place feature tp_bottom of top_plate on feature sp2_face1
                                          of side_plate_2).


Feature Level Subgoals deferred as Choice Points

(place feature tp_bottom of top_plate on feature sp2_face2
                                          of side_plate_2).


Subgoals with intial confidence rate assignments:
```

```
Feature Level Subgoals                                  Confidence
                                                           Rates

(place feature tp_bottom of top_plate on
      feature sp2_face3 of side_plate_2)                0.33333334

(place feature tp_bottom of top_plate on
      feature sp2_face1 of side_plate_2)                0.33333334

Deferred Feature Level Subgoals                         Confidence
                                                           Rates

(place feature tp_bottom of top_plate on
      feature sp2_face2 of side_plate_2)                0.33333334


Subgoal Selection:

Feature Level Subgoals                                  Confidence
                                                           Rates

(place feature tp_bottom of top_plate on
      feature sp2_face3 of side_plate_2)                0.33333334

(place feature tp_bottom of top_plate on
      feature sp2_face1 of side_plate_2)                0.33333334

Based on the confidence rates associated with feature level
subgoals
(place feature tp_bottom of top_plate on
      feature sp2_face3 of side_plate_2)
has been chosen as the hypothesized subgoal.


--------------------------------------------------


 Creating feature level subgoals for object level goal

                (place top_plate on side_plate_1) ........

                        .
                        .
                        .
```

```
Hypothesized subgoal:

(place feature tp_bottom of top_plate on feature sp1_face3
                                          of side_plate_1)

-------------------------------------------------

 Creating feature level subgoals for object level goal

                (place side_plate_2 on base_plate) ........


 After applying the constraints associated with the operations
 the feature level subgoals are:

Feature Level Subgoals

(place feature sp2_face3 of side_plate_2 on feature plate_top
                                              of base_plate).

(place feature sp2_face2 of side_plate_2 on feature plate_top
                                               of base_plate).

(place feature sp2_face1 of side_plate_2 on feature plate_top
                                               of base_plate).


Applying beta-symmetry of objects to reduce search tree ..


Applying alpha-symmetry of objects to reduce search tree..

Feature Level Subgoals

(place feature sp2_face3 of side_plate_2 on feature plate_top
                                               of base_plate).

(place feature sp2_face1 of side_plate_2 on feature plate_top
                                               of base_plate).
```

Feature Level Subgoals deferred as Choice Points

(place feature sp2_face2 of side_plate_2 on feature plate_top
of base_plate).

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(place feature sp2_face3 of side_plate_2 on feature plate_top of base_plate)	0.33333334
(place feature sp2_face1 of side_plate_2 on feature plate_top of base_plate)	0.33333334

Deferred Feature Level Subgoals	Confidence Rates
(place feature sp2_face2 of side_plate_2 on feature plate_top of base_plate)	0.33333334

Subgoal Selection:

Feature Level Subgoals	Confidence Rates
(place feature sp2_face3 of side_plate_2 on feature plate_top of base_plate)	0.33333334
(place feature sp2_face1 of side_plate_2 on feature plate_top of base_plate)	0.33333334

Based on the confidence rates associated with feature level subgoals
(place feature sp2_face3 of side_plate_2 on
feature plate_top of base_plate)
has been chosen as the hypothesized subgoal.

```
 Creating feature level subgoals for object level goal

                  (place side_plate_1 on base_plate) ........

                                    .
                                    .
                                    .
                                    .
                                    .

Hypothesized subgoal:

(place feature sp1_face3 of side_plate_1 on feature plate_top
                                                    of base_plate)

-----------------------------------------------------

 Creating feature level subgoals for object level goal

                  (insert capacitor into base_plate) ........


 After applying the constraints associated with the operations
 the feature level subgoals are:

Feature Level Subgoals

(insert feature c_side of capacitor into feature hole
                                        of base_plate).

(insert feature c_top of capacitor into feature hole
                                        of base_plate).


Applying beta-symmetry of objects to reduce search tree ..


Applying alpha-symmetry of objects to reduce search tree..


Subgoals with intial confidence rate assignments:
```

Feature Level Subgoals	Confidence Rates
(insert feature c_side of capacitor into feature hole of base_plate)	0.5
(insert feature c_top of capacitor into feature hole of base_plate)	0.5

Subgoal Selection:

Feature Level Subgoals	Confidence Rates
(insert feature c_side of capacitor into feature hole of base_plate)	0.5
(insert feature c_top of capacitor into feature hole of base_plate)	0.5

Based on the confidence rates associated with feature level subgoals
(insert feature c_side of capacitor into feature hole of base_plate)
has been chosen as the hypothesized subgoal.

Creating feature level subgoals for object level goal

(thread s_bolt4 into base_plate) .....

After applying the constraints associated with the operations the feature level subgoals are:

Feature Level Subgoals

(thread feature sb4_thread of s_bolt4 into feature sh4 of base_plate).

(thread feature sb4_thread of s_bolt4 into feature sh3
of base_plate).

(thread feature sb4_thread of s_bolt4 into feature sh2
of base_plate).

(thread feature sb4_thread of s_bolt4 into feature sh1
of base_plate).

Applying beta-symmetry of objects to reduce search tree ..

Applying alpha-symmetry of objects to reduce search tree..

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(thread feature sb4_thread of s_bolt4 into feature sh4 of base_plate)	0.25
(thread feature sb4_thread of s_bolt4 into feature sh3 of base_plate)	0.25
(thread feature sb4_thread of s_bolt4 into feature sh2 of base_plate)	0.25
(thread feature sb4_thread of s_bolt4 into feature sh1 of base_plate)	0.25

Functional symmetry:

Subgoal
(thread feature sb4_thread of s_bolt4 into feature sh3
of base_plate)
is moved to the list of choice points. There is a functionally
symmetrical alternative in
(thread feature sb4_thread of s_bolt4 into feature sh4
of base_plate).

Subgoal
(thread feature sb4_thread of s_bolt4 into feature sh2

```
                                        of base_plate)
is moved to the list of choice points. There is a functionally
symmetrical alternative in
(thread feature sb4_thread of s_bolt4 into feature sh4
                                        of base_plate).

Subgoal
(thread feature sb4_thread of s_bolt4 into feature sh1
                                         of base_plate)
is moved to the list of choice points. There is a functionally
symmetrical alternative in
(thread feature sb4_thread of s_bolt4 into feature sh4
                                        of base_plate).

Hypothesized subgoal:

(thread feature sb4_thread of s_bolt4 into feature sh4
                                        of base_plate).

-----------------------------------------------------


 Creating feature level subgoals for object level goal

                (thread s_bolt3 into base_plate) ........

                              .
                              .
                              .
                              .
                              .

Hypothesized subgoal:

(thread feature sb3_thread of s_bolt3 into feature sh4
                                        of base_plate).

-----------------------------------------------------


 Creating feature level subgoals for object level goal

                (thread s_bolt2 into base_plate) ........

                              .
```

.
.
.
.

Hypothesized subgoal:

(thread feature sb2_thread of s_bolt2 into feature sh4
of base_plate).

Creating feature level subgoals for object level goal

(thread s_bolt1 into base_plate) .....

.
.
.
.
.

Hypothesized subgoal:

(thread feature sb1_thread of s_bolt1 into feature sh4
of base_plate).

Creating feature level subgoals for object level goal

(insert s_bolt4 into transformer) .....

After applying the constraints associated with the operations
the feature level subgoals are:

Feature Level Subgoals

(insert feature sb4_shaft of s_bolt4 into feature h4

```
                                    of transformer).

(insert feature sb4_shaft of s_bolt4 into feature h3
                                    of transformer).

(insert feature sb4_shaft of s_bolt4 into feature h2
                                    of transformer).

(insert feature sb4_shaft of s_bolt4 into feature h1
                                    of transformer).


Applying beta-symmetry of objects to reduce search tree ..

Feature Level Subgoals

(insert feature sb4_shaft of s_bolt4 into feature h4
                                    of transformer).

(insert feature sb4_shaft of s_bolt4 into feature h3
                                    of transformer).


Feature Level Subgoals deferred as Choice Points

(insert feature sb4_shaft of s_bolt4 into feature h2
                                    of transformer).

(insert feature sb4_shaft of s_bolt4 into feature h1
                                    of transformer).


Applying alpha-symmetry of objects to reduce search tree..


Subgoals with intial confidence rate assignments:

Feature Level Subgoals                                  Confidence
                                                          Rates

(insert feature sb4_shaft of s_bolt4 into
               feature h4 of transformer)                 0.25

(insert feature sb4_shaft of s_bolt4 into
               feature h3 of transformer)                 0.25
```

Deferred Feature Level Subgoals — Confidence Rates

(insert feature sb4_shaft of s_bolt4 into feature h2 of transformer) — 0.25

(insert feature sb4_shaft of s_bolt4 into feature h1 of transformer) — 0.25

Functional symmetry:

Subgoal
(insert feature sb4_shaft of s_bolt4 into feature h3 of transformer)
is moved to the list of choice points. There is a functionally symmetrical alternative in
(insert feature sb4_shaft of s_bolt4 into feature h4 of transformer).

Hypothesized subgoal:

(insert feature sb4_shaft of s_bolt4 into feature h4 of transformer).

Creating feature level subgoals for object level goal

(insert s_bolt3 into transformer) .....

.
.
.
.
.

Hypothesized subgoal:

(insert feature sb3_shaft of s_bolt3 into feature h4 of transformer).

```
 Creating feature level subgoals for object level goal

                (insert s_bolt2 into transformer) ........

                              .
                              .
                              .
                              .
                              .

Hypothesized subgoal:

(insert feature sb2_shaft of s_bolt2 into feature h4
                                    of transformer).

-------------------------------------------------

 Creating feature level subgoals for object level goal

                (insert s_bolt1 into transformer) ........

                              .
                              .
                              .
                              .
                              .

Hypothesized subgoal:

(insert feature sb1_shaft of s_bolt1 into feature h4
                                    of transformer).

-------------------------------------------------

 Creating feature level subgoals for object level goal

                (place transformer on base_plate) ........
```

After applying the constraints associated with the operations the feature level subgoals are:

Feature Level Subgoals

(place feature bottom of transformer on feature plate_top of base_plate).

(place feature top of transformer on feature plate_top of base_plate).

Applying beta-symmetry of objects to reduce search tree ..

Applying alpha-symmetry of objects to reduce search tree..

Subgoals with intial confidence rate assignments:

Feature Level Subgoals	Confidence Rates
(place feature bottom of transformer on feature plate_top of base_plate)	0.5
(place feature top of transformer on feature plate_top of base_plate)	0.5

Functional symmetry:

Subgoal
(place feature top of transformer on feature plate_top of base_plate)
is moved to the list of choice points. There is a functionally symmetrical alternative in
(place feature bottom of transformer on feature plate_top of base_plate).

Hypothesized subgoal:

(place feature bottom of transformer on feature plate_top of base_plate).

Accessibility Analysis

An alternative feature subgoal
(thread feature lb3_thread of l_bolt3 into feature lh3
of base_plate)
is chosen for the object level goal
(thread l_bolt3 into base_plate).

An alternative feature subgoal
(thread feature lb2_thread of l_bolt2 into feature lh3
of base_plate)
is chosen for the object level goal
(thread l_bolt2 into base_plate).

An alternative feature subgoal
(thread feature lb2_thread of l_bolt2 into feature lh2
of base_plate)
is chosen for the object level goal
(thread l_bolt2 into base_plate).

An alternative feature subgoal
(thread feature lb1_thread of l_bolt1 into feature lh3
of base_plate)
is chosen for the object level goal
(thread l_bolt1 into base_plate).

An alternative feature subgoal
(thread feature lb1_thread of l_bolt1 into feature lh2
of base_plate)
is chosen for the object level goal
(thread l_bolt1 into base_plate).

An alternative feature subgoal
(thread feature lb1_thread of l_bolt1 into feature lh1
of base_plate)
is chosen for the object level goal
(thread l_bolt1 into base_plate).

An alternative feature subgoal
(insert feature lb3_shaft of l_bolt3 into feature sp2_h1
of side_plate_2)
is chosen for the object level goal
(insert l_bolt3 into side_plate_2).

```
An alternative feature subgoal
(insert feature lb1_shaft of l_bolt1 into feature sp1_h1
                                          of side_plate_1)
is chosen for the object level goal
(insert l_bolt1 into side_plate_1).

An alternative feature subgoal
(insert feature lb3_shaft of l_bolt3 into feature tp_h3
                                          of top_plate)
is chosen for the object level goal
(insert l_bolt3 into top_plate).

An alternative feature subgoal
(insert feature lb2_shaft of l_bolt2 into feature tp_h3
                                          of top_plate)
is chosen for the object level goal
(insert l_bolt2 into top_plate).

An alternative feature subgoal
(insert feature lb2_shaft of l_bolt2 into feature tp_h2
                                          of top_plate)
is chosen for the object level goal
(insert l_bolt2 into top_plate).

An alternative feature subgoal
(insert feature lb1_shaft of l_bolt1 into feature tp_h3
                                          of top_plate)
is chosen for the object level goal
(insert l_bolt1 into top_plate).

An alternative feature subgoal
(insert feature lb1_shaft of l_bolt1 into feature tp_h2
                                          of top_plate)
is chosen for the object level goal
(insert l_bolt1 into top_plate).

An alternative feature subgoal
(insert feature lb1_shaft of l_bolt1 into feature tp_h1
                                          of top_plate)
is chosen for the object level goal
(insert l_bolt1 into top_plate).

An alternative feature subgoal
(place feature sp2_face2 of side_plate_2 on feature plate_top
```

```
                                                    of base_plate)
is chosen for the object level goal
(place side_plate_2 on base_plate).

An alternative feature subgoal
(place feature sp1_face2 of side_plate_1 on feature plate_top
                                                    of base_plate)
is chosen for the object level goal
(place side_plate_1 on base_plate).

An alternative feature subgoal
(thread feature sb3_thread of s_bolt3 into feature sh3
                                          of base_plate)
is chosen for the object level goal
(thread s_bolt3 into base_plate).

An alternative feature subgoal
(thread feature sb2_thread of s_bolt2 into feature sh3
                                          of base_plate)
is chosen for the object level goal
(thread s_bolt2 into base_plate).

An alternative feature subgoal
(thread feature sb2_thread of s_bolt2 into feature sh2
                                          of base_plate)
is chosen for the object level goal
(thread s_bolt2 into base_plate).

An alternative feature subgoal
(thread feature sb1_thread of s_bolt1 into feature sh3
                                          of base_plate)
is chosen for the object level goal
(thread s_bolt1 into base_plate).

An alternative feature subgoal
(thread feature sb1_thread of s_bolt1 into feature sh2
                                          of base_plate)
is chosen for the object level goal
(thread s_bolt1 into base_plate).

An alternative feature subgoal
(thread feature sb1_thread of s_bolt1 into feature sh1
                                          of base_plate)
is chosen for the object level goal
(thread s_bolt1 into base_plate).
```

```
An alternative feature subgoal
(insert feature sb3_shaft of s_bolt3 into feature h2
                                     of transformer)
is chosen for the object level goal
(insert s_bolt3 into transformer).

An alternative feature subgoal
(insert feature sb2_shaft of s_bolt2 into feature h2
                                     of transformer)
is chosen for the object level goal
(insert s_bolt2 into transformer).

An alternative feature subgoal
(insert feature sb2_shaft of s_bolt2 into feature h1
                                     of transformer)
is chosen for the object level goal
(insert s_bolt2 into transformer).

An alternative feature subgoal
(insert feature sb1_shaft of s_bolt1 into feature h2
                                     of transformer)
is chosen for the object level goal
(insert s_bolt1 into transformer).

An alternative feature subgoal
(insert feature sb1_shaft of s_bolt1 into feature h1
                                     of transformer)
is chosen for the object level goal
(insert s_bolt1 into transformer).

An alternative feature subgoal
(insert feature sb1_shaft of s_bolt1 into feature h3
                                     of transformer)
is chosen for the object level goal
(insert s_bolt1 into transformer).


--------------------------------------------------
          Tentative Plan at the Feature Level
--------------------------------------------------

(thread feature lb4_thread of l_bolt4 into feature lh4
                                       of base_plate).
```

```
(thread feature lb3_thread of l_bolt3 into feature lh3
                                        of base_plate).

(thread feature lb2_thread of l_bolt2 into feature lh2
                                        of base_plate).

(thread feature lb1_thread of l_bolt1 into feature lh1
                                        of base_plate).

(insert feature lb4_shaft of l_bolt4 into feature sp2_h4
                                        of side_plate_2).

(insert feature lb3_shaft of l_bolt3 into feature sp2_h1
                                        of side_plate_2).

(insert feature lb2_shaft of l_bolt2 into feature sp1_h4
                                        of side_plate_1).

(insert feature lb1_shaft of l_bolt1 into feature sp1_h1
                                        of side_plate_1).

(insert feature lb4_shaft of l_bolt4 into feature tp_h4
                                        of top_plate).

(insert feature lb3_shaft of l_bolt3 into feature tp_h3
                                        of top_plate).

(insert feature lb2_shaft of l_bolt2 into feature tp_h2
                                        of top_plate).

(insert feature lb1_shaft of l_bolt1 into feature tp_h1
                                        of top_plate).

(place feature tp_bottom of top_plate on feature sp2_face3
                                        of side_plate_2).

(place feature tp_bottom of top_plate on feature sp1_face3
                                        of side_plate_1).

(place feature sp2_face2 of side_plate_2 on feature plate_top
                                                of base_plate).

(place feature sp1_face2 of side_plate_1 on feature plate_top
                                                of base_plate).
```

```
(insert feature c_side of capacitor into feature hole
                                        of base_plate).

(thread feature sb4_thread of s_bolt4 into feature sh4
                                        of base_plate).

(thread feature sb3_thread of s_bolt3 into feature sh3
                                        of base_plate).

(thread feature sb2_thread of s_bolt2 into feature sh2
                                        of base_plate).

(thread feature sb1_thread of s_bolt1 into feature sh1
                                        of base_plate).

(insert feature sb4_shaft of s_bolt4 into feature h4
                                      of transformer).

(insert feature sb3_shaft of s_bolt3 into feature h2
                                      of transformer).

(insert feature sb2_shaft of s_bolt2 into feature h1
                                      of transformer).

(insert feature sb1_shaft of s_bolt1 into feature h3
                                      of transformer).

(place feature bottom of transformer on feature plate_top
                                          of base_plate).
```

E.3 RAPT Input Generated by ASSEMBLE

As specified in Section 6.6.2, ASSEMBLE generates subfeatures corresponding to the mating features, the requisite information for processing by RAPT including spatial relationships between subfeatures, and produces a file comprising a set of prolog assumptions. In the case of the controller assembly experiment, the file generated by ASSEMBLE is comprised of 340 declarations. As this appeared to be too large for RAPT to handle, the set of declarations were divided into two subsets: one corresponding to the subassembly of the base_plate, transformer, capacitor and the four small bolts; and the other

corresponding to the subassembly of the base_plate, two side_plates, top_plate and the four large bolts. These subassemblies are referred to as subassembly-1 and subassembly-2. Goals 1 through 10 make up subassembly-1 and goals 11 through 16 correspond to subassembly-2. The input files presented to RAPT corresponding to the two subassemblies of the controller assembly experiment are presented below. The ouput generated by ASSEMBLE is the union of both of these sets of declarations.

E.3.1 Input to RAPT Corresponding to Subassembly-1

```
:-init(assume).

;;; declarations pertaining to subfeatures of hole in base_plate.

:-assume(hole_curved_face$base_plate:cylindrical_face).
:-assume(r$hole_curved_face$base_plate=2.8).
:-assume(loc$hole_curved_face$base_plate=Pos_98),
        Pos_98 is stpos(0,0,1, 1,0,0, 0,1,0, 4.95,1.15,-1.0).
:-assume(hole_botttom_face$base_plate:plane_face).
:-assume(loc$hole_botttom_face$base_plate=Pos_99),
        Pos_99 is stpos(0,0,1, 1,0,0, 0,1,0, 4.95,1.15,-1.0).

;;; declarations pertaining to subfeatures of c_side in capacitor.

:-assume(c_side_curved_face$capacitor:cylindrical_face).
:-assume(r$c_side_curved_face$capacitor=2.5).
:-assume(loc$c_side_curved_face$capacitor=Pos_100),
        Pos_100 is stpos(-1,0,0, 0,1,0, 0,0,-1, -5.6,0,0).
:-assume(c_side_botttom_face$capacitor:plane_face).
:-assume(loc$c_side_botttom_face$capacitor=Pos_101),
        Pos_101 is stpos(-1,0,0, 0,1,0, 0,0,-1, -5.6,0,0).

;;; spatial relations between base_plate and capacitor.

:-assume(c_side_botttom_face$capacitor against
                              hole_botttom_face$base_plate).
:-assume(c_side_curved_face$capacitor fits
                             hole_curved_face$base_plate).

;;; declarations pertaining to subfeatures of sh4 in base_plate.

:-assume(sh4_curved_face$base_plate:cylindrical_face).
:-assume(r$sh4_curved_face$base_plate=0.25).
```

```
:-assume(loc$sh4_curved_face$base_plate=Pos_104),
        Pos_104 is stpos(0,0,1, 1,0,0, 0,1,0, -2.4,4.8,0.7).
:-assume(sh4_botttom_face$base_plate:plane_face).
:-assume(loc$sh4_botttom_face$base_plate=Pos_105),
        Pos_105 is stpos(0,0,1, 1,0,0, 0,1,0, -2.4,4.8,0.7).

;;; declarations pertaining to subfeatures of s_bolt4.

:-assume(sb4_thread_curved_face$s_bolt4:cylindrical_face).
:-assume(r$sb4_thread_curved_face$s_bolt4=0.25).
:-assume(loc$sb4_thread_curved_face$s_bolt4=Pos_106),
        Pos_106 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).
:-assume(sb4_thread_botttom_face$s_bolt4:plane_face).
:-assume(loc$sb4_thread_botttom_face$s_bolt4=Pos_107),
        Pos_107 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).

;;; spatial relations between base_plate and s_bolt4.

:-assume(sb4_thread_botttom_face$s_bolt4 against
                                    sh4_botttom_face$base_plate).
:-assume(sb4_thread_curved_face$s_bolt4 fits
                                   sh4_curved_face$base_plate).

;;; declarations pertaining to subfeatures of sh3 in base_plate.

:-assume(sh3_curved_face$base_plate:cylindrical_face).
:-assume(r$sh3_curved_face$base_plate=0.25).
:-assume(loc$sh3_curved_face$base_plate=Pos_110),
        Pos_110 is stpos(0,0,1, 1,0,0, 0,1,0, -2.4,-4.8,0.7).
:-assume(sh3_botttom_face$base_plate:plane_face).
:-assume(loc$sh3_botttom_face$base_plate=Pos_111),
        Pos_111 is stpos(0,0,1, 1,0,0, 0,1,0, -2.4,-4.8,0.7).

;;; declarations pertaining to subfeatures of s_bolt3.

:-assume(sb3_thread_curved_face$s_bolt3:cylindrical_face).
:-assume(r$sb3_thread_curved_face$s_bolt3=0.25).
:-assume(loc$sb3_thread_curved_face$s_bolt3=Pos_112),
        Pos_112 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).
:-assume(sb3_thread_botttom_face$s_bolt3:plane_face).
:-assume(loc$sb3_thread_botttom_face$s_bolt3=Pos_113),
        Pos_113 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).

;;; spatial relations between base_plate and s_bolt3.
```

```
:-assume(sb3_thread_botttom_face$s_bolt3 against
                                    sh3_botttom_face$base_plate).
:-assume(sb3_thread_curved_face$s_bolt3 fits
                                   sh3_curved_face$base_plate).

;;; declarations pertaining to subfeatures of sh2 in base_plate.

:-assume(sh2_curved_face$base_plate:cylindrical_face).
:-assume(r$sh2_curved_face$base_plate=0.25).
:-assume(loc$sh2_curved_face$base_plate=Pos_116),
       Pos_116 is stpos(0,0,1, 1,0,0, 0,1,0, -8.1,4.8,0.7).
:-assume(sh2_botttom_face$base_plate:plane_face).
:-assume(loc$sh2_botttom_face$base_plate=Pos_117),
       Pos_117 is stpos(0,0,1, 1,0,0, 0,1,0, -8.1,4.8,0.7).

;;; declarations pertaining to subfeatures of s_bolt2.

:-assume(sb2_thread_curved_face$s_bolt2:cylindrical_face).
:-assume(r$sb2_thread_curved_face$s_bolt2=0.25).
:-assume(loc$sb2_thread_curved_face$s_bolt2=Pos_118),
       Pos_118 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).
:-assume(sb2_thread_botttom_face$s_bolt2:plane_face).
:-assume(loc$sb2_thread_botttom_face$s_bolt2=Pos_119),
       Pos_119 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).

;;; spatial relations between base_plate and s_bolt2.

:-assume(sb2_thread_botttom_face$s_bolt2 against
                                    sh2_botttom_face$base_plate).
:-assume(sb2_thread_curved_face$s_bolt2 fits
                                   sh2_curved_face$base_plate).

;;; declarations pertaining to subfeatures of sh1 in base_plate.

:-assume(sh1_curved_face$base_plate:cylindrical_face).
:-assume(r$sh1_curved_face$base_plate=0.25).
:-assume(loc$sh1_curved_face$base_plate=Pos_122),
       Pos_122 is stpos(0,0,1, 1,0,0, 0,1,0, -8.1,-4.8,0.7).
:-assume(sh1_botttom_face$base_plate:plane_face).
:-assume(loc$sh1_botttom_face$base_plate=Pos_123),
       Pos_123 is stpos(0,0,1, 1,0,0, 0,1,0, -8.1,-4.8,0.7).

;;; declarations pertaining to subfeatures of s_bolt1.

:-assume(sb1_thread_curved_face$s_bolt1:cylindrical_face).
```

```
:-assume(r$sb1_thread_curved_face$s_bolt1=0.25).
:-assume(loc$sb1_thread_curved_face$s_bolt1=Pos_124),
        Pos_124 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).
:-assume(sb1_thread_botttom_face$s_bolt1:plane_face).
:-assume(loc$sb1_thread_botttom_face$s_bolt1=Pos_125),
        Pos_125 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).

;;; spatial relations between base_plate and s_bolt1.

:-assume(sb1_thread_botttom_face$s_bolt1 against
                                    sh1_botttom_face$base_plate).
:-assume(sb1_thread_curved_face$s_bolt1 fits
                                   sh1_curved_face$base_plate).

;;; declarations pertaining to subfeatures of h4 in transformer.

:-assume(h4_curved_face$transformer:cylindrical_face).
:-assume(r$h4_curved_face$transformer=0.25).
:-assume(loc$h4_curved_face$transformer=Pos_126),
        Pos_126 is stpos(1,0,0, 0,1,0, 0,0,1, -5.2,-4.8,2.85).
:-assume(h4_botttom_face$transformer:plane_face).
:-assume(loc$h4_botttom_face$transformer=Pos_127),
        Pos_127 is stpos(1,0,0, 0,1,0, 0,0,1, -5.2,-4.8,2.85).

;;; declarations pertaining to subfeatures of s_bolt4.

:-assume(sb4_shaft_curved_face$s_bolt4:cylindrical_face).
:-assume(r$sb4_shaft_curved_face$s_bolt4=0.25).
:-assume(loc$sb4_shaft_curved_face$s_bolt4=Pos_128),
        Pos_128 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).
:-assume(sb4_shaft_botttom_face$s_bolt4:plane_face).
:-assume(loc$sb4_shaft_botttom_face$s_bolt4=Pos_129),
        Pos_129 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).

;;; spatial relations between transformer and s_bolt4.

:-assume(sb4_shaft_botttom_face$s_bolt4 against
                                    h4_botttom_face$transformer).
:-assume(sb4_shaft_curved_face$s_bolt4 fits
                                   h4_curved_face$transformer).

;;; declarations pertaining to subfeatures of h2 in transformer.

:-assume(h2_curved_face$transformer:cylindrical_face).
:-assume(r$h2_curved_face$transformer=0.25).
```

```
:-assume(loc$h2_curved_face$transformer=Pos_130),
        Pos_130 is stpos(1,0,0, 0,1,0, 0,0,1, -5.2,4.8,-2.85).
:-assume(h2_botttom_face$transformer:plane_face).
:-assume(loc$h2_botttom_face$transformer=Pos_131),
        Pos_131 is stpos(1,0,0, 0,1,0, 0,0,1, -5.2,4.8,-2.85).

;;; declarations pertaining to subfeatures of s_bolt3.

:-assume(sb3_shaft_curved_face$s_bolt3:cylindrical_face).
:-assume(r$sb3_shaft_curved_face$s_bolt3=0.25).
:-assume(loc$sb3_shaft_curved_face$s_bolt3=Pos_132),
        Pos_132 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).
:-assume(sb3_shaft_botttom_face$s_bolt3:plane_face).
:-assume(loc$sb3_shaft_botttom_face$s_bolt3=Pos_133),
        Pos_133 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).

;;; spatial relations between transformer and s_bolt3.

:-assume(sb3_shaft_botttom_face$s_bolt3 against
                                  h2_botttom_face$transformer).
:-assume(sb3_shaft_curved_face$s_bolt3 fits
                                 h2_curved_face$transformer).

;;; declarations pertaining to subfeatures of h1 in transformer.

:-assume(h1_curved_face$transformer:cylindrical_face).
:-assume(r$h1_curved_face$transformer=0.25).
:-assume(loc$h1_curved_face$transformer=Pos_134),
        Pos_134 is stpos(1,0,0, 0,1,0, 0,0,1, -5.2,-4.8,-2.85).
:-assume(h1_botttom_face$transformer:plane_face).
:-assume(loc$h1_botttom_face$transformer=Pos_135),
        Pos_135 is stpos(1,0,0,.0,1,0, 0,0,1, -5.2,-4.8,-2.85).

;;; declarations pertaining to subfeatures of s_bolt2.

:-assume(sb2_shaft_curved_face$s_bolt2:cylindrical_face).
:-assume(r$sb2_shaft_curved_face$s_bolt2=0.25).
:-assume(loc$sb2_shaft_curved_face$s_bolt2=Pos_136),
        Pos_136 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).
:-assume(sb2_shaft_botttom_face$s_bolt2:plane_face).
:-assume(loc$sb2_shaft_botttom_face$s_bolt2=Pos_137),
        Pos_137 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).

;;; spatial relations between transformer and s_bolt2.
```

```
:-assume(sb2_shaft_botttom_face$s_bolt2 against
                                    h1_botttom_face$transformer).
:-assume(sb2_shaft_curved_face$s_bolt2 fits
                                   h1_curved_face$transformer).

;;; declarations pertaining to subfeatures of h3 in transformer.

:-assume(h3_curved_face$transformer:cylindrical_face).
:-assume(r$h3_curved_face$transformer=0.25).
:-assume(loc$h3_curved_face$transformer=Pos_138),
        Pos_138 is stpos(1,0,0, 0,1,0, 0,0,1, -5.2,4.8,2.85).
:-assume(h3_botttom_face$transformer:plane_face).
:-assume(loc$h3_botttom_face$transformer=Pos_139),
        Pos_139 is stpos(1,0,0, 0,1,0, 0,0,1, -5.2,4.8,2.85).

;;; declarations pertaining to subfeatures of s_bolt1.

:-assume(sb1_shaft_curved_face$s_bolt1:cylindrical_face).
:-assume(r$sb1_shaft_curved_face$s_bolt1=0.25).
:-assume(loc$sb1_shaft_curved_face$s_bolt1=Pos_140),
        Pos_140 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).
:-assume(sb1_shaft_botttom_face$s_bolt1:plane_face).
:-assume(loc$sb1_shaft_botttom_face$s_bolt1=Pos_141),
        Pos_141 is stpos(-1,0,0, 0,1,0, 0,0,-1, -0.65,0,0).

;;; spatial relations between transformer and s_bolt1.

:-assume(sb1_shaft_botttom_face$s_bolt1 against
                                    h3_botttom_face$transformer).
:-assume(sb1_shaft_curved_face$s_bolt1 fits
                                   h3_curved_face$transformer).

;;; declarations pertaining to subfeature of feature plate_top of
;;; base_plate.

:-assume(plate_top_plane$base_plate:plane_face).
:-assume(loc$plate_top_plane$base_plate=Pos_94),
        Pos_94 is stpos(0,0,1, 1,0,0, 0,1,0, 0,0,1).

;;; declarations pertaing to subfeature of feature bottom of
;;; transformer.

:-assume(bottom_plane$transformer:plane_face).
:-assume(loc$bottom_plane$transformer=Pos_143),
        Pos_143 is stpos(-1,0,0, 0,1,0, 0,0,-1, -5.2,0,0).
```

```
;;; spatial relations between transformer and base_plate.

:-assume(bottom_plane$transformer against
                                    plate_top_plane$base_plate).

;;; absolute location of base_plate.

:-assume(loc$base_plate=trans(0,0,0)).
```

E.3.2 Input to RAPT Corresponding to Subassembly-2

```
:-init(assume).

;;; declarations pertaining to subfeatures of lh4 in base_plate.

:-assume(lh4_curved_face$base_plate:cylindrical_face).
:-assume(r$lh4_curved_face$base_plate=0.6).
:-assume(loc$lh4_curved_face$base_plate=Pos_36),
       Pos_36 is stpos(0,0,1, 1,0,0, 0,1,0, -5.6,-6.8,-1.0).
:-assume(lh4_botttom_face$base_plate:plane_face).
:-assume(loc$lh4_botttom_face$base_plate=Pos_37),
       Pos_37 is stpos(0,0,1, 1,0,0, 0,1,0, -5.6,-6.8,-1.0).

;;; declarations pertaining to subfeatures of l_bolt4.

:-assume(lb4_thread_curved_face$l_bolt4:cylindrical_face).
:-assume(r$lb4_thread_curved_face$l_bolt4=0.6).
:-assume(loc$lb4_thread_curved_face$l_bolt4=Pos_38),
       Pos_38 is stpos(-1,0,0, 0,1,0, 0,0,-1, -6.25,0,0).
:-assume(lb4_thread_botttom_face$l_bolt4:plane_face).
:-assume(loc$lb4_thread_botttom_face$l_bolt4=Pos_39),
       Pos_39 is stpos(-1,0,0, 0,1,0, 0,0,-1, -6.25,0,0).

;;; spatial relationships between base_plate and l_bolt4.

:-assume(lb4_thread_botttom_face$l_bolt4 against
                                     lh4_botttom_face$base_plate).
:-assume(lb4_thread_curved_face$l_bolt4 fits
                                    lh4_curved_face$base_plate).

;;; declarations pertaining to subfeatures of lh3 in base_plate.
```

```
:-assume(lh3_curved_face$base_plate:cylindrical_face).
:-assume(r$lh3_curved_face$base_plate=0.6).
:-assume(loc$lh3_curved_face$base_plate=Pos_42),
        Pos_42 is stpos(0,0,1, 1,0,0, 0,1,0, -5.6,6.8,-1.0).
:-assume(lh3_botttom_face$base_plate:plane_face).
:-assume(loc$lh3_botttom_face$base_plate=Pos_43),
        Pos_43 is stpos(0,0,1, 1,0,0, 0,1,0, -5.6,6.8,-1.0).

;;; declarations pertaining to subfeatures of l_bolt3.

:-assume(lb3_thread_curved_face$l_bolt3:cylindrical_face).
:-assume(r$lb3_thread_curved_face$l_bolt3=0.6).
:-assume(loc$lb3_thread_curved_face$l_bolt3=Pos_44),
        Pos_44 is stpos(-1,0,0, 0,1,0, 0,0,-1, -6.25,0,0).
:-assume(lb3_thread_botttom_face$l_bolt3:plane_face).
:-assume(loc$lb3_thread_botttom_face$l_bolt3=Pos_45),
        Pos_45 is stpos(-1,0,0, 0,1,0, 0,0,-1, -6.25,0,0).

;;; spatial relationships between base_plate and l_bolt3.

:-assume(lb3_thread_botttom_face$l_bolt3 against
                                    lh3_botttom_face$base_plate).
:-assume(lb3_thread_curved_face$l_bolt3 fits
                                   lh3_curved_face$base_plate).

;;; declarations pertaining to subfeatures of lh2 in base_plate.

:-assume(lh2_curved_face$base_plate:cylindrical_face).
:-assume(r$lh2_curved_face$base_plate=0.6).
:-assume(loc$lh2_curved_face$base_plate=Pos_48),
        Pos_48 is stpos(0,0,1, 1,0,0, 0,1,0, 5.6,-6.8,-1.0).
:-assume(lh2_botttom_face$base_plate:plane_face).
:-assume(loc$lh2_botttom_face$base_plate=Pos_49),
        Pos_49 is stpos(0,0,1, 1,0,0, 0,1,0, 5.6,-6.8,-1.0).

;;; declarations pertaining to subfeatures of l_bolt2.

:-assume(lb2_thread_curved_face$l_bolt2:cylindrical_face).
:-assume(r$lb2_thread_curved_face$l_bolt2=0.6).
:-assume(loc$lb2_thread_curved_face$l_bolt2=Pos_50),
        Pos_50 is stpos(-1,0,0, 0,1,0, 0,0,-1, -6.25,0,0).
:-assume(lb2_thread_botttom_face$l_bolt2:plane_face).
:-assume(loc$lb2_thread_botttom_face$l_bolt2=Pos_51),
        Pos_51 is stpos(-1,0,0, 0,1,0, 0,0,-1, -6.25,0,0).
```

```
;;; spatial relationships between base_plate and l_bolt2.

:-assume(lb2_thread_botttom_face$l_bolt2 against
                                    lh2_botttom_face$base_plate).
:-assume(lb2_thread_curved_face$l_bolt2 fits
                                   lh2_curved_face$base_plate).

;;; declarations pertaining to subfeatures of lh1 in base_plate.

:-assume(lh1_curved_face$base_plate:cylindrical_face).
:-assume(r$lh1_curved_face$base_plate=0.6).
:-assume(loc$lh1_curved_face$base_plate=Pos_54),
       Pos_54 is stpos(0,0,1, 1,0,0, 0,1,0, 5.6,6.8,-1.0).
:-assume(lh1_botttom_face$base_plate:plane_face).
:-assume(loc$lh1_botttom_face$base_plate=Pos_55),
       Pos_55 is stpos(0,0,1, 1,0,0, 0,1,0, 5.6,6.8,-1.0).

;;; declarations pertaining to subfeatures of l_bolt1.

:-assume(lb1_thread_curved_face$l_bolt1:cylindrical_face).
:-assume(r$lb1_thread_curved_face$l_bolt1=0.6).
:-assume(loc$lb1_thread_curved_face$l_bolt1=Pos_56),
       Pos_56 is stpos(-1,0,0, 0,1,0, 0,0,-1, -6.25,0,0).
:-assume(lb1_thread_botttom_face$l_bolt1:plane_face).
:-assume(loc$lb1_thread_botttom_face$l_bolt1=Pos_57),
       Pos_57 is stpos(-1,0,0, 0,1,0, 0,0,-1, -6.25,0,0).

;;; spatial relationships between base_plate and l_bolt1.

:-assume(lb1_thread_botttom_face$l_bolt1 against
                                    lh1_botttom_face$base_plate).
:-assume(lb1_thread_curved_face$l_bolt1 fits
                                   lh1_curved_face$base_plate).

;;; declarations pertaining to subfeatures of sp2_h4.

:-assume(sp2_h4_curved_face$side_plate_2:cylindrical_face).
:-assume(r$sp2_h4_curved_face$side_plate_2=0.6).
:-assume(loc$sp2_h4_curved_face$side_plate_2=Pos_58),
       Pos_58 is stpos(0,1,0, 1,0,0, 0,0,-1, 5.6,-5.1,0.0).
:-assume(sp2_h4_botttom_face$side_plate_2:plane_face).
:-assume(loc$sp2_h4_botttom_face$side_plate_2=Pos_59),
       Pos_59 is stpos(0,1,0, 1,0,0, 0,0,-1, 5.6,-5.1,0.0).

;;; declarations pertaining to subfeatures of l_bolt4.
```

```
:-assume(lb4_shaft_curved_face$l_bolt4:cylindrical_face).
:-assume(r$lb4_shaft_curved_face$l_bolt4=0.6).
:-assume(loc$lb4_shaft_curved_face$l_bolt4=Pos_60),
        Pos_60 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0).
:-assume(lb4_shaft_botttom_face$l_bolt4:plane_face).
:-assume(loc$lb4_shaft_botttom_face$l_bolt4=Pos_61),
        Pos_61 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0).

;;; spatial relations between l_bolt4 and side_plate_2.

:-assume(lb4_shaft_botttom_face$l_bolt4 against
                                sp2_h4_botttom_face$side_plate_2).
:-assume(lb4_shaft_curved_face$l_bolt4 fits
                                sp2_h4_curved_face$side_plate_2).

;;; declarations pertaining to subfeatures of sp2_h1.

:-assume(sp2_h1_curved_face$side_plate_2:cylindrical_face).
:-assume(r$sp2_h1_curved_face$side_plate_2=0.6).
:-assume(loc$sp2_h1_curved_face$side_plate_2=Pos_62),
        Pos_62 is stpos(0,-1,0, 1,0,0, 0,0,1, -5.6,5.1,0.0).
:-assume(sp2_h1_botttom_face$side_plate_2:plane_face).
:-assume(loc$sp2_h1_botttom_face$side_plate_2=Pos_63),
        Pos_63 is stpos(0,-1,0, 1,0,0, 0,0,1, -5.6,5.1,0.0).

;;; declarations pertaining to subfeatures of l_bolt3.

:-assume(lb3_shaft_curved_face$l_bolt3:cylindrical_face).
:-assume(r$lb3_shaft_curved_face$l_bolt3=0.6).
:-assume(loc$lb3_shaft_curved_face$l_bolt3=Pos_64),
        Pos_64 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0).
:-assume(lb3_shaft_botttom_face$l_bolt3:plane_face).
:-assume(loc$lb3_shaft_botttom_face$l_bolt3=Pos_65),
        Pos_65 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0).

;;; spatial relations between l_bolt3 and side_plate_2.

:-assume(lb3_shaft_botttom_face$l_bolt3 against
                                sp2_h1_botttom_face$side_plate_2).
:-assume(lb3_shaft_curved_face$l_bolt3 fits
                                sp2_h1_curved_face$side_plate_2).

;;; declarations pertaining to subfeatures of sp1_h4.
```

```
:-assume(sp1_h4_curved_face$side_plate_1:cylindrical_face).
:-assume(r$sp1_h4_curved_face$side_plate_1=0.6).
:-assume(loc$sp1_h4_curved_face$side_plate_1=Pos_66),
        Pos_66 is stpos(0,1,0, 1,0,0, 0,0,-1, 5.6,-5.1,0.0).
:-assume(sp1_h4_botttom_face$side_plate_1:plane_face).
:-assume(loc$sp1_h4_botttom_face$side_plate_1=Pos_67),
        Pos_67 is stpos(0,1,0, 1,0,0, 0,0,-1, 5.6,-5.1,0.0).

;;; declarations pertaining to subfeatures of l_bolt2.

:-assume(lb2_shaft_curved_face$l_bolt2:cylindrical_face).
:-assume(r$lb2_shaft_curved_face$l_bolt2=0.6).
:-assume(loc$lb2_shaft_curved_face$l_bolt2=Pos_68),
        Pos_68 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0).
:-assume(lb2_shaft_botttom_face$l_bolt2:plane_face).
:-assume(loc$lb2_shaft_botttom_face$l_bolt2=Pos_69),
        Pos_69 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0).

;;; spatial relations between l_bolt2 and side_plate_1.

:-assume(lb2_shaft_botttom_face$l_bolt2 against
                                sp1_h4_botttom_face$side_plate_1).
:-assume(lb2_shaft_curved_face$l_bolt2 fits
                                sp1_h4_curved_face$side_plate_1).

;;; declarations pertaining to subfeatures of sp1_h1.

:-assume(sp1_h1_curved_face$side_plate_1:cylindrical_face).
:-assume(r$sp1_h1_curved_face$side_plate_1=0.6).
:-assume(loc$sp1_h1_curved_face$side_plate_1=Pos_70),
        Pos_70 is stpos(0,-1,0, 1,0,0, 0,0,1, -5.6,5.1,0.0).
:-assume(sp1_h1_botttom_face$side_plate_1:plane_face).
:-assume(loc$sp1_h1_botttom_face$side_plate_1=Pos_71),
        Pos_71 is stpos(0,-1,0, 1,0,0, 0,0,1, -5.6,5.1,0.0).

;;; declarations pertaining to subfeatures of l_bolt1.

:-assume(lb1_shaft_curved_face$l_bolt1:cylindrical_face).
:-assume(r$lb1_shaft_curved_face$l_bolt1=0.6).
:-assume(loc$lb1_shaft_curved_face$l_bolt1=Pos_72),
        Pos_72 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0).
:-assume(lb1_shaft_botttom_face$l_bolt1:plane_face).
:-assume(loc$lb1_shaft_botttom_face$l_bolt1=Pos_73),
        Pos_73 is stpos(-1,0,0, 0,1,0, 0,0,-1, -2.45,0,0).
```

```
;;; spatial relations between l_bolt1 and side_plate_1.

:-assume(lb1_shaft_botttom_face$l_bolt1 against
                                  sp1_h1_botttom_face$side_plate_1).
:-assume(lb1_shaft_curved_face$l_bolt1 fits
                                 sp1_h1_curved_face$side_plate_1).

;;; declarations pertaining to subfeatures of tp_h4.

:-assume(tp_h4_curved_face$top_plate:cylindrical_face).
:-assume(r$tp_h4_curved_face$top_plate=0.6).
:-assume(loc$tp_h4_curved_face$top_plate=Pos_74),
       Pos_74 is stpos(0,0,1, 1,0,0, 0,1,0, 5.5,6.5,-0.2).
:-assume(tp_h4_botttom_face$top_plate:plane_face).
:-assume(loc$tp_h4_botttom_face$top_plate=Pos_75),
       Pos_75 is stpos(0,0,1, 1,0,0, 0,1,0, 5.5,6.5,-0.2).

;;; spatial relations between l_bolt4 and top_plate.

:-assume(lb4_shaft_botttom_face$l_bolt4 against
                                    tp_h4_botttom_face$top_plate).
:-assume(lb4_shaft_curved_face$l_bolt4 fits
                                   tp_h4_curved_face$top_plate).

;;; declarations pertaining to subfeatures of tp_h3.

:-assume(tp_h3_curved_face$top_plate:cylindrical_face).
:-assume(r$tp_h3_curved_face$top_plate=0.6).
:-assume(loc$tp_h3_curved_face$top_plate=Pos_78),
       Pos_78 is stpos(0,0,1, 1,0,0, 0,1,0, -5.5,6.5,-0.2).
:-assume(tp_h3_botttom_face$top_plate:plane_face).
:-assume(loc$tp_h3_botttom_face$top_plate=Pos_79),
       Pos_79 is stpos(0,0,1, 1,0,0, 0,1,0, -5.5,6.5,-0.2).

;;; spatial relations between l_bolt3 and top_plate.

:-assume(lb3_shaft_botttom_face$l_bolt3 against
                                    tp_h3_botttom_face$top_plate).
:-assume(lb3_shaft_curved_face$l_bolt3 fits
                                   tp_h3_curved_face$top_plate).

;;; declarations pertaining to subfeatures of tp_h2.

:-assume(tp_h2_curved_face$top_plate:cylindrical_face).
:-assume(r$tp_h2_curved_face$top_plate=0.6).
```

```
:-assume(loc$tp_h2_curved_face$top_plate=Pos_82),
       Pos_82 is stpos(0,0,1, 1,0,0, 0,1,0, 5.5,-6.5,-0.2).
:-assume(tp_h2_botttom_face$top_plate:plane_face).
:-assume(loc$tp_h2_botttom_face$top_plate=Pos_83),
       Pos_83 is stpos(0,0,1, 1,0,0, 0,1,0, 5.5,-6.5,-0.2).

;;; spatial relations between l_bolt2 and top_plate.

:-assume(lb2_shaft_botttom_face$l_bolt2 against
                                  tp_h2_botttom_face$top_plate).
:-assume(lb2_shaft_curved_face$l_bolt2 fits
                                 tp_h2_curved_face$top_plate).

;;; declarations pertaining to subfeatures of tp_h1.

:-assume(tp_h1_curved_face$top_plate:cylindrical_face).
:-assume(r$tp_h1_curved_face$top_plate=0.6).
:-assume(loc$tp_h1_curved_face$top_plate=Pos_86),
       Pos_86 is stpos(0,0,1, 1,0,0, 0,1,0, -5.5,-6.5,0.0).
:-assume(tp_h1_botttom_face$top_plate:plane_face).
:-assume(loc$tp_h1_botttom_face$top_plate=Pos_87),
       Pos_87 is stpos(0,0,1, 1,0,0, 0,1,0, -5.5,-6.5,0.0).

;;; spatial relations between l_bolt1 and top_plate.

:-assume(lb1_shaft_botttom_face$l_bolt1 against
                                  tp_h1_botttom_face$top_plate).
:-assume(lb1_shaft_curved_face$l_bolt1 fits
                                 tp_h1_curved_face$top_plate).

;;; declarations pertaining to subfeature of sp2_face3.

:-assume(sp2_face3_plane$side_plate_2:plane_face).
:-assume(loc$sp2_face3_plane$side_plate_2=Pos_90),
       Pos_90 is stpos(0,1,0, 1,0,0, 0,0,-1, 0,5.1,0).

;;; declarations pertaining to subfeature of tp_bottom.

:-assume(tp_bottom_plane$top_plate:plane_face).
:-assume(loc$tp_bottom_plane$top_plate=Pos_91),
       Pos_91 is stpos(0,0,-1, 1,0,0, 0,-1,0, 0,0,-1.0).

;;; spatial relationship between top_plate and side_plate_2.

:-assume(tp_bottom_plane$top_plate against
```

```
                                sp2_face3_plane$side_plate_2).

;;; declarations pertaining to subfeature of sp1_face3.

:-assume(sp1_face3_plane$side_plate_1:plane_face).
:-assume(loc$sp1_face3_plane$side_plate_1=Pos_92),
        Pos_92 is stpos(0,1,0, 1,0,0, 0,0,-1, 0,5.1,0).

;;; spatial relationship between top_plate and side_plate_1.

:-assume(tp_bottom_plane$top_plate against
                               sp1_face3_plane$side_plate_1).

;;; declarations pertaining to subfeature of plate_top.

:-assume(plate_top_plane$base_plate:plane_face).
:-assume(loc$plate_top_plane$base_plate=Pos_94),
        Pos_94 is stpos(0,0,1, 1,0,0, 0,1,0, 0,0,1).

;;; declarations pertaining to subfeature of sp2_face2.

:-assume(sp2_face2_plane$side_plate_2:plane_face).
:-assume(loc$sp2_face2_plane$side_plate_2=Pos_95),
        Pos_95 is stpos(0,-1,0, 1,0,0, 0,0,1, 0,-5.1,0).

;;; spatial relationship between base_plate and side_plate_2.

:-assume(sp2_face2_plane$side_plate_2 against
                               plate_top_plane$base_plate).

;;; declarations pertaining to subfeature of sp1_face2.

:-assume(sp1_face2_plane$side_plate_1:plane_face).
:-assume(loc$sp1_face2_plane$side_plate_1=Pos_97),
        Pos_97 is stpos(0,-1,0, 1,0,0, 0,0,1, 0,-5.1,0).

;;; spatial relationship between base_plate and side_plate_1.

:-assume(sp1_face2_plane$side_plate_1 against
                               plate_top_plane$base_plate).

;;; absolute location of base_plate.

:-assume(loc$base_plate=trans(0,0,0)).
```

E.4 Output Produced by RAPT for Subassembly-1

```
The following relationships and locations will be used:
bottom_plane$transformer against plate_top_plane$base_plate
c_side_botttom_face$capacitor against
                                    hole_botttom_face$base_plate
sb1_shaft_botttom_face$s_bolt1 against
                                     h3_botttom_face$transformer
sb1_thread_bottom_face$s_bolt1 against
                                     sh1_bottom_face$base_plate
sb2_shaft_botttom_face$s_bolt2 against
                                     h1_botttom_face$transformer
sb2_thread_bottom_face$s_bolt2 against
                                     sh2_bottom_face$base_plate
sb3_shaft_botttom_face$s_bolt3 against
                                     h2_botttom_face$transformer
sb3_thread_bottom_face$s_bolt3 against
                                     sh3_bottom_face$base_plate
sb4_shaft_botttom_face$s_bolt4 against
                                     h4_botttom_face$transformer
sb4_thread_bottom_face$s_bolt4 against
                                     sh4_bottom_face$base_plate
c_side_curved_face$capacitor fits hole_curved_face$base_plate
sb1_shaft_curved_face$s_bolt1 fits h3_curved_face$transformer
sb1_thread_curved_face$s_bolt1 fits sh1_curved_face$base_plate
sb2_shaft_curved_face$s_bolt2 fits h1_curved_face$transformer
sb2_thread_curved_face$s_bolt2 fits sh2_curved_face$base_plate
sb3_shaft_curved_face$s_bolt3 fits h2_curved_face$transformer
sb3_thread_curved_face$s_bolt3 fits sh3_curved_face$base_plate
sb4_shaft_curved_face$s_bolt4 fits h4_curved_face$transformer
sb4_thread_curved_face$s_bolt4 fits sh4_curved_face$base_plate
loc$base_plate = pos(1,0,0; 0,1,0; 0,0,1; 0,0,0)

 Message from RAPT:


The following relationships and locations will be used:
bottom_plane$transformer against plate_top_plane$base_plate
c_side_botttom_face$capacitor against hole_botttom_face$base_plate
sb1_shaft_botttom_face$s_bolt1 against h3_botttom_face$transformer
sb1_thread_bottom_face$s_bolt1 against sh1_bottom_face$base_plate
sb2_shaft_botttom_face$s_bolt2 against h1_botttom_face$transformer
sb2_thread_bottom_face$s_bolt2 against sh2_bottom_face$base_plate
sb3_shaft_botttom_face$s_bolt3 against h2_botttom_face$transformer
sb3_thread_bottom_face$s_bolt3 against sh3_bottom_face$base_plate
```

```
sb4_shaft_botttom_face$s_bolt4 against h4_botttom_face$transformer
sb4_thread_bottom_face$s_bolt4 against sh4_bottom_face$base_plate
c_side_curved_face$capacitor fits hole_curved_face$base_plate
sb1_shaft_curved_face$s_bolt1 fits h3_curved_face$transformer
sb1_thread_curved_face$s_bolt1 fits sh1_curved_face$base_plate
sb2_shaft_curved_face$s_bolt2 fits h1_curved_face$transformer
sb2_thread_curved_face$s_bolt2 fits sh2_curved_face$base_plate
sb3_shaft_curved_face$s_bolt3 fits h2_curved_face$transformer
sb3_thread_curved_face$s_bolt3 fits sh3_curved_face$base_plate
sb4_shaft_curved_face$s_bolt4 fits h4_curved_face$transformer
sb4_thread_curved_face$s_bolt4 fits sh4_curved_face$base_plate

Points of action of the rotations inconsistant.
Separation distances different.
Distance 1 : 9.6
Distance 2 : 11.1647

Points of action of the rotations inconsistant.
Separation distances different.
Distance 1 : 0.0
Distance 2 : 5.7

Points of action of the rotations inconsistant.
Separation distances different.
Distance 1 : 0.0
Distance 2 : 5.7


The following locations have been inferred:


 Message from RAPT:

                .
                .
                .

 Message from RAPT:

                .
                .
                .

 Message from RAPT:
```

```
.
.
.
```

Message from RAPT:

```
.
.
.
```

Message from RAPT:

```
The following relationships and locations will be used:
bottom_plane$transformer against plate_top_plane$base_plate
c_side_botttom_face$capacitor against hole_botttom_face$base_plate
sb1_shaft_botttom_face$s_bolt1 against h3_botttom_face$transformer
sb1_thread_bottom_face$s_bolt1 against sh1_bottom_face$base_plate
sb2_shaft_botttom_face$s_bolt2 against h1_botttom_face$transformer
sb2_thread_bottom_face$s_bolt2 against sh2_bottom_face$base_plate
sb3_shaft_botttom_face$s_bolt3 against h2_botttom_face$transformer
sb3_thread_bottom_face$s_bolt3 against sh3_bottom_face$base_plate
sb4_shaft_botttom_face$s_bolt4 against h4_botttom_face$transformer
sb4_thread_bottom_face$s_bolt4 against sh4_bottom_face$base_plate
c_side_curved_face$capacitor fits hole_curved_face$base_plate
sb1_shaft_curved_face$s_bolt1 fits h3_curved_face$transformer
sb1_thread_curved_face$s_bolt1 fits sh1_curved_face$base_plate
sb2_shaft_curved_face$s_bolt2 fits h1_curved_face$transformer
sb2_thread_curved_face$s_bolt2 fits sh2_curved_face$base_plate
sb3_shaft_curved_face$s_bolt3 fits h2_curved_face$transformer
sb3_thread_curved_face$s_bolt3 fits sh3_curved_face$base_plate
sb4_shaft_curved_face$s_bolt4 fits h4_curved_face$transformer
sb4_thread_curved_face$s_bolt4 fits sh4_curved_face$base_plate
loc$transformer =
     pos(0.0,0.0,1.0; 0.0,-1.0,0.0; 1.0,0.0,0.0; -5.25,0.0,5.9)
loc$base_plate = pos(1,0,0; 0,1,0; 0,0,1; 0,0,0)
loc$capacitor = pos(0,0,1; 1,0,0; 0,1,0; 4.95,1.15,4.6)
loc$s_bolt1 = pos(0,0,1; 1,0,0; 0,1,0; -8.1,-4.8,1.35)
loc$s_bolt2 = pos(0,0,1; 1,0,0; 0,1,0; -8.1,4.8,1.35)
loc$s_bolt3 = pos(0,0,1; 1,0,0; 0,1,0; -2.4,-4.8,1.35)
loc$s_bolt4 = pos(0,0,1; 1,0,0; 0,1,0; -2.4,4.8,1.35)
;;; DECLARING VARIABLE loc$capacitor
;;; DECLARING VARIABLE loc$s_bolt1
;;; DECLARING VARIABLE loc$s_bolt2
;;; DECLARING VARIABLE loc$s_bolt3
```

```
;;; DECLARING VARIABLE loc$s_bolt4
;;; DECLARING VARIABLE loc$transformer
```

E.5 Output Produced by RAPT for Subassembly-2

```
The following relationships and locations will be used:
lb1_shaft_botttom_face$l_bolt1 against
                               sp1_h1_botttom_face$side_plate_1
lb1_shaft_botttom_face$l_bolt1 against tp_h1_botttom_face$top_plate
lb1_thread_bottom_face$l_bolt1 against lh1_bottom_face$base_plate
lb2_shaft_botttom_face$l_bolt2 against
                               sp1_h4_botttom_face$side_plate_1
lb2_shaft_botttom_face$l_bolt2 against tp_h2_botttom_face$top_plate
lb2_thread_bottom_face$l_bolt2 against lh2_bottom_face$base_plate
lb3_shaft_botttom_face$l_bolt3 against
                               sp2_h1_botttom_face$side_plate_2
lb3_shaft_botttom_face$l_bolt3 against tp_h3_botttom_face$top_plate
lb3_thread_bottom_face$l_bolt3 against lh3_bottom_face$base_plate
lb4_shaft_botttom_face$l_bolt4 against
                               sp2_h4_botttom_face$side_plate_2
lb4_shaft_botttom_face$l_bolt4 against tp_h4_botttom_face$top_plate
lb4_thread_bottom_face$l_bolt4 against lh4_bottom_face$base_plate
sp1_face2_plane$side_plate_1 against plate_top_plane$base_plate
sp2_face2_plane$side_plate_2 against plate_top_plane$base_plate
tp_bottom_plane$top_plate against sp1_face3_plane$side_plate_1
tp_bottom_plane$top_plate against sp2_face3_plane$side_plate_2
lb1_shaft_curved_face$l_bolt1 fits sp1_h1_curved_face$side_plate_1
lb1_shaft_curved_face$l_bolt1 fits tp_h1_curved_face$top_plate
lb1_thread_curved_face$l_bolt1 fits lh1_curved_face$base_plate
lb2_shaft_curved_face$l_bolt2 fits sp1_h4_curved_face$side_plate_1
lb2_shaft_curved_face$l_bolt2 fits tp_h2_curved_face$top_plate
lb2_thread_curved_face$l_bolt2 fits lh2_curved_face$base_plate
lb3_shaft_curved_face$l_bolt3 fits sp2_h1_curved_face$side_plate_2
lb3_shaft_curved_face$l_bolt3 fits tp_h3_curved_face$top_plate
lb3_thread_curved_face$l_bolt3 fits lh3_curved_face$base_plate
lb4_shaft_curved_face$l_bolt4 fits sp2_h4_curved_face$side_plate_2
lb4_shaft_curved_face$l_bolt4 fits tp_h4_curved_face$top_plate
lb4_thread_curved_face$l_bolt4 fits lh4_curved_face$base_plate
loc$base_plate = pos(1,0,0; 0,1,0; 0,0,1; 0,0,0)

 Message from RAPT:
```

```
The following relationships and locations will be used:
lb1_shaft_botttom_face$l_bolt1 against
                                sp1_h1_botttom_face$side_plate_1
lb1_shaft_botttom_face$l_bolt1 against tp_h1_botttom_face$top_plate
lb1_thread_bottom_face$l_bolt1 against lh1_bottom_face$base_plate
lb2_shaft_botttom_face$l_bolt2 against
                                sp1_h4_botttom_face$side_plate_1
lb2_shaft_botttom_face$l_bolt2 against tp_h2_botttom_face$top_plate
lb2_thread_bottom_face$l_bolt2 against lh2_bottom_face$base_plate
lb3_shaft_botttom_face$l_bolt3 against
                                sp2_h1_botttom_face$side_plate_2
lb3_shaft_botttom_face$l_bolt3 against tp_h3_botttom_face$top_plate
lb3_thread_bottom_face$l_bolt3 against lh3_bottom_face$base_plate
lb4_shaft_botttom_face$l_bolt4 against
                                sp2_h4_botttom_face$side_plate_2
lb4_shaft_botttom_face$l_bolt4 against tp_h4_botttom_face$top_plate
lb4_thread_bottom_face$l_bolt4 against lh4_bottom_face$base_plate
sp1_face2_plane$side_plate_1 against plate_top_plane$base_plate
sp2_face2_plane$side_plate_2 against plate_top_plane$base_plate
tp_bottom_plane$top_plate against sp1_face3_plane$side_plate_1
tp_bottom_plane$top_plate against sp2_face3_plane$side_plate_2
lb1_shaft_curved_face$l_bolt1 fits sp1_h1_curved_face$side_plate_1
lb1_shaft_curved_face$l_bolt1 fits tp_h1_curved_face$top_plate
lb1_thread_curved_face$l_bolt1 fits lh1_curved_face$base_plate
lb2_shaft_curved_face$l_bolt2 fits sp1_h4_curved_face$side_plate_1
lb2_shaft_curved_face$l_bolt2 fits tp_h2_curved_face$top_plate
lb2_thread_curved_face$l_bolt2 fits lh2_curved_face$base_plate
lb3_shaft_curved_face$l_bolt3 fits sp2_h1_curved_face$side_plate_2
lb3_shaft_curved_face$l_bolt3 fits tp_h3_curved_face$top_plate
lb3_thread_curved_face$l_bolt3 fits lh3_curved_face$base_plate
lb4_shaft_curved_face$l_bolt4 fits sp2_h4_curved_face$side_plate_2
lb4_shaft_curved_face$l_bolt4 fits tp_h4_curved_face$top_plate
lb4_thread_curved_face$l_bolt4 fits lh4_curved_face$base_plate

Angles between pairs of features not equal
Angle1 : 180.0 Angle2 : 0.0
Relation 1 was derived from :
Consequence : 154
Consequence : 158
Consequence : 161
Consequence : 160
Consequence : 159
Consequence : 157
Consequence : 103
```

```
Consequence : 111
Consequence : 112
Consequence : 100
Consequence : 119
Consequence : 170
Consequence : 127
Consequence : 128
Consequence : 169
Consequence : 116
Relation 2 was derived from :
Consequence : 168
Consequence : 170
Consequence : 171
Consequence : 172
Consequence : 169
Consequence : 165
Perhaps you have confused against/coplanar or fits/against?
May I invert the second relation and continue?

Angles between pairs of features not equal
Angle1 : 180.0 Angle2 : 0.0
Relation 1 was derived from :
Consequence : 116
Consequence : 128
Consequence : 127
Consequence : 119
Consequence : 168
Consequence : 170
Consequence : 171
Consequence : 172
Consequence : 169
Consequence : 165
Relation 2 was derived from :
Consequence : 100
Consequence : 112
Consequence : 111
Consequence : 103
Consequence : 157
Consequence : 159
Consequence : 160
Consequence : 161
Consequence : 158
Consequence : 154
Perhaps you have confused against/coplanar or fits/against?
May I invert the second relation and continue?
```

```
Angles between pairs of features not equal
Angle1 : 0.0 Angle2 : 180.0
Relation 1 was derived from :
Consequence : 165
Consequence : 169
Consequence : 172
Consequence : 171
Consequence : 170
Consequence : 168
Consequence : 100
Consequence : 112
Consequence : 111
Consequence : 103
Consequence : 157
Consequence : 159
Consequence : 160
Consequence : 161
Consequence : 158
Consequence : 154
Relation 2 was derived from :
Consequence : 119
Consequence : 170
Consequence : 127
Consequence : 128
Consequence : 169
Consequence : 116
Perhaps you have confused against/coplanar or fits/against?
May I invert the second relation and continue?

Distance from feature to plane not equal for each pair of features
Distance1 : -10.2 Distance2 : -1.0

Distance from feature to plane not equal for each pair of features
Distance1 : -1.0 Distance2 : -10.2

Distance from feature to plane not equal for each pair of features
Distance1 : 0.0 Distance2 : 1.8

                    .
                    .
                    .
 Message from RAPT:
```

```
The following relationships and locations will be used:
lb1_shaft_botttom_face $ l_bolt1 against
                         sp1_h1_botttom_face $ side_plate_1
lb1_shaft_botttom_face $ l_bolt1 against
                         tp_h1_botttom_face $ top_plate
lb1_thread_bottom_face $ l_bolt1 against
                         lh1_bottom_face $ base_plate
lb2_shaft_botttom_face $ l_bolt2 against
                         sp1_h4_botttom_face $ side_plate_1
lb2_shaft_botttom_face $ l_bolt2 against
                         tp_h2_botttom_face $ top_plate
lb2_thread_bottom_face $ l_bolt2 against
                         lh2_bottom_face $ base_plate
lb3_shaft_botttom_face $ l_bolt3 against
                         sp2_h1_botttom_face $ side_plate_2
lb3_shaft_botttom_face $ l_bolt3 against
                         tp_h3_botttom_face $ top_plate
lb3_thread_bottom_face $ l_bolt3 against
                         lh3_bottom_face $ base_plate
lb4_shaft_botttom_face $ l_bolt4 against
                         sp2_h4_botttom_face $ side_plate_2
lb4_shaft_botttom_face $ l_bolt4 against
                         tp_h4_botttom_face $ top_plate
lb4_thread_bottom_face $ l_bolt4 against
                         lh4_bottom_face $ base_plate
sp1_face2_plane $ side_plate_1 against
                              plate_top_plane $ base_plate
sp2_face2_plane $ side_plate_2 against
                              plate_top_plane $ base_plate
tp_bottom_plane $ top_plate against
                            sp1_face3_plane $ side_plate_1
tp_bottom_plane $ top_plate against
                            sp2_face3_plane $ side_plate_2
lb1_shaft_curved_face $ l_bolt1 fits
                        sp1_h1_curved_face $ side_plate_1
lb1_shaft_curved_face $ l_bolt1 fits
                        tp_h1_curved_face $ top_plate
lb1_thread_curved_face $ l_bolt1 fits
                         lh1_curved_face $ base_plate
lb2_shaft_curved_face $ l_bolt2 fits
                        sp1_h4_curved_face $ side_plate_1
lb2_shaft_curved_face $ l_bolt2 fits
                        tp_h2_curved_face $ top_plate
lb2_thread_curved_face $ l_bolt2 fits
                         lh2_curved_face $ base_plate
```

```
lb3_shaft_curved_face $ l_bolt3 fits
                        sp2_h1_curved_face $ side_plate_2
lb3_shaft_curved_face $ l_bolt3 fits tp_h3_curved_face $ top_plate
lb3_thread_curved_face $ l_bolt3 fits lh3_curved_face $ base_plate
lb4_shaft_curved_face $ l_bolt4 fits
                                sp2_h4_curved_face $ side_plate_2
lb4_shaft_curved_face $ l_bolt4 fits tp_h4_curved_face $ top_plate
lb4_thread_curved_face $ l_bolt4 fits lh4_curved_face $ base_plate
loc $ l_bolt1 = pos(0,0,1; 1,0,0; 0,1,0; 5.6,6.8,5.25)
loc $ side_plate_1 = pos(1,0,0; 0,0,1; 0,-1,0; 0.0,-6.8,7.9)
loc $ top_plate = pos(1,0,0; 0,1,0; 0,0,1; 0.1,-0.299999,3.0)
loc $ base_plate = pos(1,0,0; 0,1,0; 0,0,1; 0,0,0)
loc $ l_bolt2 = pos(0,0,1; 1,0,0; 0,1,0; 5.6,-6.8,5.25)
loc $ l_bolt3 = pos(0,0,1; 1,0,0; 0,1,0; -5.6,6.8,5.25)
loc $ side_plate_2 = pos(1,0,0; 0,0,1; 0,-1,0; -11.2,-6.8,7.9)
loc $ l_bolt4 = pos(0,0,1; 1,0,0; 0,1,0; -5.6,-6.8,5.25)
```

BIBLIOGRAPHY

[1] Agre, P.E. and D. Chapman, "Pengi: An Implementation of a Theory of Activity," *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA, pp.268–272.

[2] Ambler, A.P., "Robotics and Solid Modelling: A Discussion of the Requirements Robotic Applications put on Solid Modelling Systems," *Proceedings of the Second International Symposium on Robotics Research*, 1984.

[3] Albus, J., MacLean, C., Barbera, A., and Fitzgerald, M., "Hierarchical Control for Robots in an Automated Factory," *Proceedings, 13th ISIR*, Chicago, Illinois, April 1983, pp.13.29–13.43.

[4] Arbib, M.A., Iberall, T., and Lyons, D., "Coordinated Control Programs for Movements of the Hand," In: *Hand Function and the Neocortex* (A.W. Goodwin and I. Darian-Smith, Eds.), *Experimental Brain Research Supplement 10*, pp.111–129.

[5] Begej, S., "A Tactile Sensing System and Optical Tactile-Sensor Array for Robotic Applications," Technical Report 85-06, Laboratory for Perceptual Robotics, Computer and Information Science Department, University of Massachusetts, Amherst, 1985.

[6] Boothroyd, G., C. Poli and L.E. Murch., *Automatic Assembly*, Marcel Dekker, Inc., New York, 1982.

[7] Brooks, R.A., "Symbolic Reasoning Among 3-D Models and 2-D Images," *Artificial Intelligence*, Vol. 17, pp.285–348, 1981.

[8] Brooks, R.A., "Symbolic Error Analysis and Robot Planning," *The International Journal of Robotics Research*, Vol. 1, pp.29–68, 1982.

[9] Brooks, R.A. and T. Lozano-Perez., "A Subdivision Algorithm In Configuration Space for Findpath with Rotation," *IJCAI 8*, Karlsruhe, Germany, 1983.

[10] Brooks, R.A., "Solving the Find-Path Problem by Good Representation of Free Space," *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13, pp.190–197, 1983.

[11] Brooks, R.A., "Planning Collision Free Motions for Pick and Place Operations," *First International Symposium on Robotics Research*, pp.5–37, August 1983.

[12] Brown, C.M., "PADL-2: A Technical Summary," *IEEE Computer Graphics and Applications*, Vol.2, No.2, pp.69–84, March 1982.

[13] CAM-I's Illustrated Glossary of Workpiece Form Features, Revised May, 1981, R-80-PPP-02.1, CAM-I, Inc., Arlington, Texas.

[14] Cameron, S. and Aylett, J., "ROBMOD: A Geometry Engine for Robotics," *Proceedings 1988 International Conference on Robotics and Automation*, Philadelphia, Pa., pp.880–885, April 1988.

[15] Corner, D.F., A.P. Ambler and R.J. Popplestone, "Reasoning about the Spatial Relationships Derived from a RAPT Program for Describing Assembly by Robot," *IJCAI 83*, pp. 842–844.

[16] Dufay, B. and J.C. Latombe, "An Approach to Automatic Robot Programming based on Inductive Learning," *First International Symposium on Robotics Research*, pp.97–115, August 1983.

[17] Erman, L.D., F. Hayes-Roth, V.R. Lesser and D.R. Reddy, "The HEARSAY-II Speech Understanding System: Integrating knowledge to resolve uncertainty," *Computing Surveys* 12, pp.213–253, 1980.

[18] Ernest, G. and A. Newell, *GPS: A Case Study in Generality and Problem Solving*, Academic Press, New York, NY, 1969.

[19] S.E. Fahlman., "A Planning System for Robot Construction Tasks," *Artificial Intelligence*, Vol. 6, No. 2, 1975.

[20] Fikes, R.E. and N.J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol. 3, pp.251–208, 1971.

[21] Fikes, R.E., P. Hart and N.J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence 3*, 4(1972),251–288.

[22] Hayes-Roth, B. and Hayes-Roth, F., "A Cognitive Model of Planning," *Cognitive Science*, 3, pp.275–310, 1979.

[23] Laugier, C. "A Program for automatic grasping of objects with a robot arm," *Eleventh International Symposium on Industrial Robots*, Tokyo, Japan, October 1981.

[24] Laugier, C. and J. Pertin, "Automatic Grasping: A Case Study in Accessibility Analysis," *International meeting on Advanced Software in Robotics*, Liege, May 1983.

[25] Laugier, C. and J. Pertin, "SHARP: A System for Automatic Programming of Manipulation robots," *Third International Symposium on Robotics Research*, October 1985.

[26] Lee, K., and D.C. Gossard, "A hierarchical data structure for representing assemblies: part 1," *Computer-aided design*, Volume 17, Number 1, Jan/Feb 1985.

[27] Lee, K. and G. Andrews, "Inference of the positions of components in an assembly: part 2," *Computer-aided design*, Volume 17, Number 1, Jan/Feb 1985.

[28] Lieberman, L.I., M.A. Wesley, "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly," *IBM Journal of Research and Development*, Volume 21, Number 4, 321–333, 1977.

[29] Liu, Y. and M.A. Arbib, "A Robot Planner in the Assembly Domain," COINS Technical Report 86-36, Department of Computer and Information Science, University of Massachusetts, Amherst, July 1986.

[30] Lozano-Perez, T., "The Design of a Mechanical Assembly System," MIT Artificial Intelligence Laboratory, TR 397, December 1976.

[31] Lozano-Perez, T., and Winston, P.H., "LAMA: A Language for Automatic Mechanical Assembly," Proceedings of the *Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, August, pp.710–716, 1977.

[32] Lozano-Perez, T., "Automatic Planning of Manipulator Transfer Movements," *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11, 681–698, 1981.

[33] Lozano-Perez, T., "Task Planning" in *Robot Motion: Planning and Control*, M.Brady *et al.* Eds. Cambridge, MA: MIT Press, 1983.

[34] Lozano-Perez, T., Mason, M.T., and Taylor, R.H., "Automatic Synthesis of Fine-Motion Strategies for Robots," *International Journal of Robotics Research*, Vol. 3, No. 1, 1984.

[35] Lozano-Perez, T. and R.A. Brooks, "An Approach to Automatic Robot Programming," A.I. Memo 842, MIT Artificial Intelligence Laboratory, April 1985.

[36] Lozano-Perez, T., *et al.*, "Handey: A Robot System that Recognizes, Plans, and Manipulates," *IEEE International Conference on Robotics and Automation*, Raleigh, North Carolina, March 31 - April 3, 1987.

[37] Lyons, D.M. "RS: A Formal Model of Distributed Computation for Sensory-Based Robot Control", Ph.D. Dissertation, appears as COINS Technical Report 86-43, Laboratory for Perceptual Robotics, Department of Computer and Information Science, University of Massachusetts, Amherst, September 1986.

[38] Mason, M.T., "Automatic planning of Fine Motions: Correctness and Completeness," *IEEE International Conference on Robotics*, Atlanta, Georgia, 1984.

[39] Miller, G.A., Galanter, E.G. and K.H. Pribram, *Plans and the Structure of Behavior*, Henry Holt and Company, New York, 1960.

[40] Morris, G.H., and L.S. Haynes, "Robotic Assembly by Constraints," *Proceedings 1987 IEEE International Conference on Robotics and Automation*, March 31-April 3, Raleigh, North Carolina, 1507–1515.

[41] Nevatia, R., "Structured descriptions of complex curved objects for recognition and visual memory. AIM-250, Stanford AI Lab, 1974.

[42] Nevins, J.L. and D.E. Whitney, *Automatica*, Volume 16, Number 6, 1980.

[43] Overton, K.J., "The Acquisition, Processing, and Use of Tactile Sensor Data in Robot Control". Ph.D. Thesis, COINS Technical Report 84-08, University of Massachusetts, Amherst, 1984.

[44] Paul, R.P., *Robot Manipulators: Mathematics, Programming and Control*, The MIT Press, Cambridge, Massachusetts, 1981.

[45] Popplestone, R.J., "RAPT: A Language for Describing Assemblies," Technical Report, University of Edinburgh, 1979.

[46] Popplestone, R.J., A.P. Ambler and I. Bellos, "An Interpreter for a Language for Describing Assemblies," *Artificial Intelligence*, 14, pp.79–107, 1980.

[47] Popplestone, R.J., Ambler, A.P., and Bellos, I.M., "An Efficient and Portable Implementation of RAPT," *Proceedings of the 1st ICAA*, Bedford, UK: IFS Publications, pp.411–422, March 1980.

[48] Popplestone, R.J., "An Integrated Design System for Enginnering," *Robotics Research: The Third International Symposium*, eds. Faugeras, O.D. and Giralt, G. MIT Press, 1985.

[49] A.G. Requicha and H.B. Voelcker, "Solid Modelling: Current Status and Research Directions," *IEEE Computer Graphics and Applications*, October 1983.

[50] Sacerdoti, E.D., "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, 5, 115–135, 1974.

[51] Sacerdoti, E.D., *A Structure for Plans and Behavior*, New York:Elsevier North-Holland, 1977.

[52] Sanderson, A.C. and L.S. Homem-de-Mello, "Task Planning and Control Synthesis for Flexible Assembly Systems," NATO International Advanced Research Workshop on Machine Intelligence and Knowledge Engineering for Robotic Applications, Maratea, Italy, May 12–16, 1986.

[53] Stefik, M., "Planning with Constraints (MOLGEN: Part 1)," *Artificial Intelligence*, 6, 111–140, 1981.

[54] Stefik, M., "Planning and Meta-Planning (MOLGEN: Part 2)," *Artificial Intelligence*, 6, 141–170, 1981.

[55] Sussman, G.J., *A Model of Skill Acquisition*, Ph.D. Thesis, MIT. (published by New York: American Elsevier Company, 1976).

[56] Tate, A., "Interacting Goals and their Use," *IJCAI 4*, pp.215–218, 1975.

[57] Tate, A., "Generating Project Networks. Proceedings of the *Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Mass., August 1977.

[58] Taylor, R.H., "The Synthesis of Manipulator Control Programs from Task-level Specifications," Stanford Artificial Intelligence Laboratory, AIM-282, July 1976.

[59] Torras, C. and F. Thomas, "Planning with Constraints: Application to Sensor-Based Robot Assembly Tasks," Proceedings of the IFAC *Symposium on Robot Control*, November 6–8, Barcelona, Spain, pp.453–456, 1985.

[60] Udupa, S.M., "Collision Detection and Avoidance in Computer Controlled Manipulators," Proceedings of the *Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, August, 737–748, 1977.

[61] Vijaykumar, R., M.A. Arbib, and Y. Liu. "Dynamic Planning for Sensor-Based Robots," Proceedings of the IFAC *Symposium on Robot Control*, November 6–8, Barcelona, Spain, pp.401–406, 1985.

[62] Waldinger, R., "Achieving Several Goals Simultaneously," *Machine Intelligence 8*, Ellis Horwood Ltd., pp.94–136, 1977.

[63] Wesley, M.A., Lozano-Perez, T., Lieberman, L.I., Lavin, M.A. and Grossman, D.D., "A Geometric Modelling System for Automated Mechanical Assembly," *IBM Journal of research and Development*, Vol. 24, No. 1, 64–74, 1980.

[64] Wesson, R.B., "Planning in the World of the Air Traffic Controller," *IJCAI 5*, pp.473–479, 1977.

[65] Wilkins, D., "Representation in a Domain-Independent Planner," *IJCAI 8*, 2, pp.733–740, 1983.