A ROBOT PLANNER

IN THE ASSEMBLY DOMAIN

Yanxi Liu

Michael A. Arbib

COINS Technical Report 86-36

July 1986

Laboratory for Perceptual robotics

Department of Computer and Information Science

A305, Graduate Research Center

University of Massachusetts

Amherst, Massachusetts 01003

This research was supported in part by a grant from the National Science Foundation, number IST-8513989

ABSTRACT

This paper describes a sensor-based robot planner for the assembly domain. The representation and control structure of the planner are discussed in detail. MOLGEN's hierarchical planning approach and least commitment strategy are adapted. Representation for each object is implemented as a hierarchical graph with features as nodes and relationships between features as arcs. Formation and propagation of the constraints are discussed in terms of relationships among features.

1. Introduction

Planning is a technique widely used in problem solving, and an important aspect of robot control. Planning is defined [1] as: *Deciding on a course of action before acting.* This definition is true for a static planner, however it is not wide enough to include a *dynamic* planner.

A *dynamic* planner is an *adaptive* planner. It can change a plan when the plan is made and is being executed, i.e. it interleaves planning and execution [2]. It is useful to change a plan because of the dynamic environment, since the robot is not the only actor changing the world. Changing circumstances could include damaged parts and unexpected obstacles being in the way. The sensing ability of the robot enables the possibility of using an adaptive planner. Only recently, the planning of assembly operations — the integration of strategic planning, geometric modeling of solids and spatial relationships, the functional use of sensors, and reduction of positioning uncertainty — has become an explicit research objective [2,3]; although some successful work has been done for each of these problems.

The final goal of this work is to integrate the strategic, geometric and functional aspects of the robot assembly problem. As a first step towards building a dynamic planner, the work described here is an off-line planner for a sensor-based robot to perform assembly tasks.This planner[1], written in ART [4] can handle several different kinds of assembly examples. It is based on the hierarchical planner structure of MOLGEN [6]. The planner has three levels: supervisor, planner and assembler. These levels will be further described in the next section.

The main purpose of this work is to find out:

1. What kind of information about assembly parts is relevant to the assembly domain and how to represent that information effectively;

[1]Right now it is changing from ART to Common Lisp.

2. What kind of knowledge of planning and assembly, represented as heuristics and constraints, is needed to make the planner control more efficient;

3. How to combine the information from 1. and 2. together to reach the goal of successfully integrating strategic, geometric and functional aspects of the robot assembly problem.

The rest of this paper is organized as follows. Section 2 is an overview of the planner including planning levels, object representations and the rule-base. Section 3 briefly surveys the history of the planner, which went through six versions. In section 4 we discuss the representation, the control and the combination of control strategy and representation hierarchy. Finally, in section 5 we conclude by answering the above questions and pointing out our possible future work.

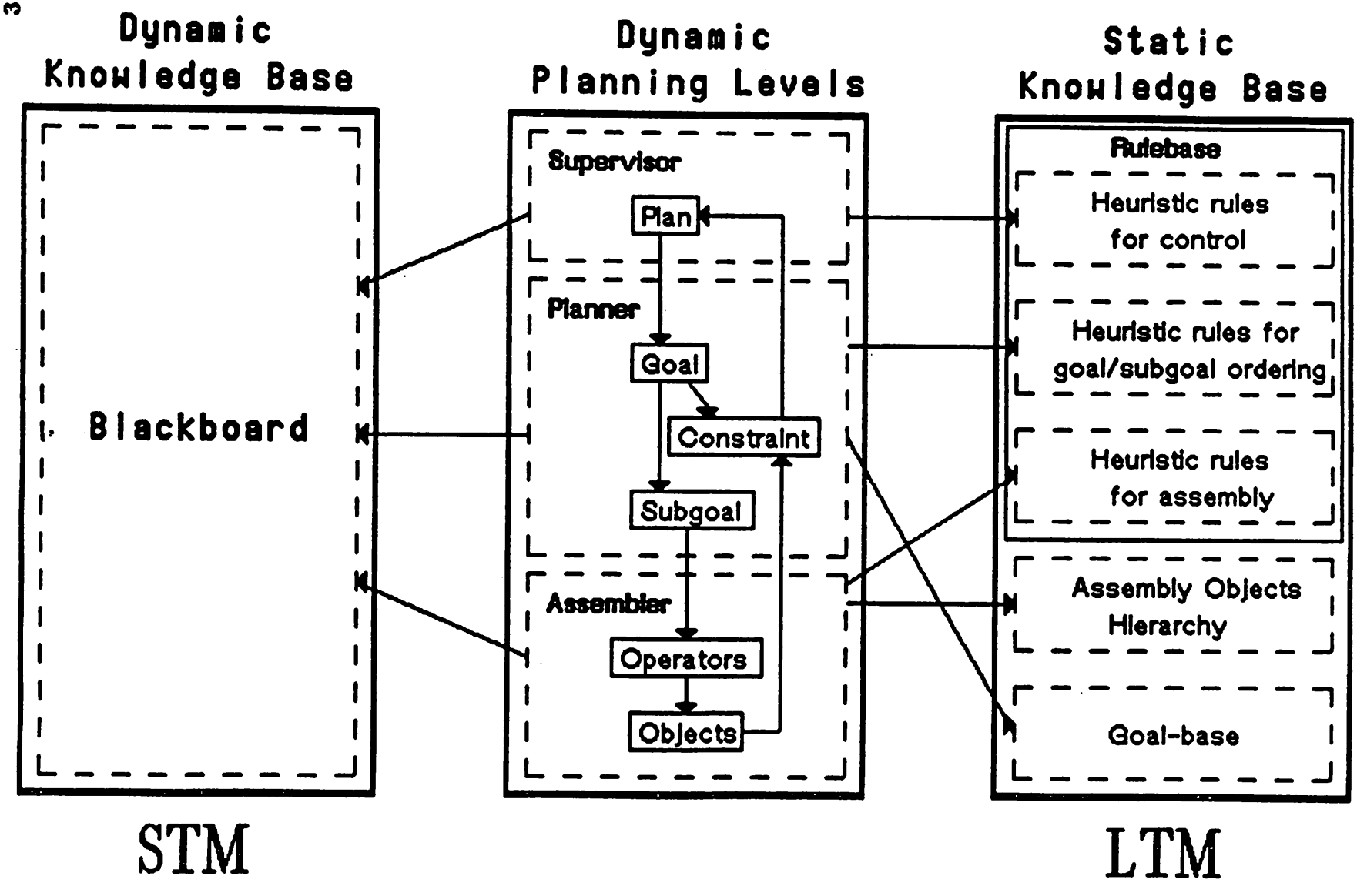


Figure 1: An overview of the planner system

2. Overview of the static planner

Figure 1 shows an overview of the planner. This overview picture will be discussed in this section in terms of planning levels, representation, and heuristic rules.

2.1 Planning levels

The planner follows the meta-planning idea from MOLGEN[8].

> Plans are constructed in one layer. decision about the design of the plan are made in a higher layer, and strategies that dictate the design decisions are made at a still higher level.[1]

MOLGEN was developed for designing experiments in molecular genetics. The robot planner is working in the assembly domain. We decided to using MOLGEN's planning structure due to the thought of simplifying the complicated problem by separating the decisions about the problem domain and the decisions about the plan process.

The planner conceptually has three levels to work on: supervisor, planner and assembler, as in Table 1.

Table 1

levels	objects	operators
supervisor	plan	guess least commitment
planner	goals subgoals constraints	refiner shifter formation propagation
assembler	parts ends features	fit thread align axis match features check if engaged

The top level, the supervisor level corresponding to MOLGEN's *strategy space*, is concerned with focus-of-attention issues as in MOLGEN. The planner begins by using the *least commitment* operator at the supervisor level, meaning that the planner will not make any decisions at any level concerning goal orders or feature matching until it has a reason to do so. When the planner has exhausted its heuristics it will turn on the *guess* operator. This will allow it to guess at a decision in order to see what is going to happen.

At the next level, the planner level corresponding to MOLGEN's *design space*, a goal is refined by the *refiner* operator into subgoals and a subgoal is reached by calling operators at the lower assembler level. The *shifter* operator changes the pointer among the subgoals.

At the assembler level, corresponding to MOLGEN's *planning space* the assembly parts are examined according to the current focused subgoal which was determined by the planner level. The operator reports its result by posting a sign on a blackboard[2] and the planner level uses this to decide what to do next.

[2]A global dynamic database, it will be explained later

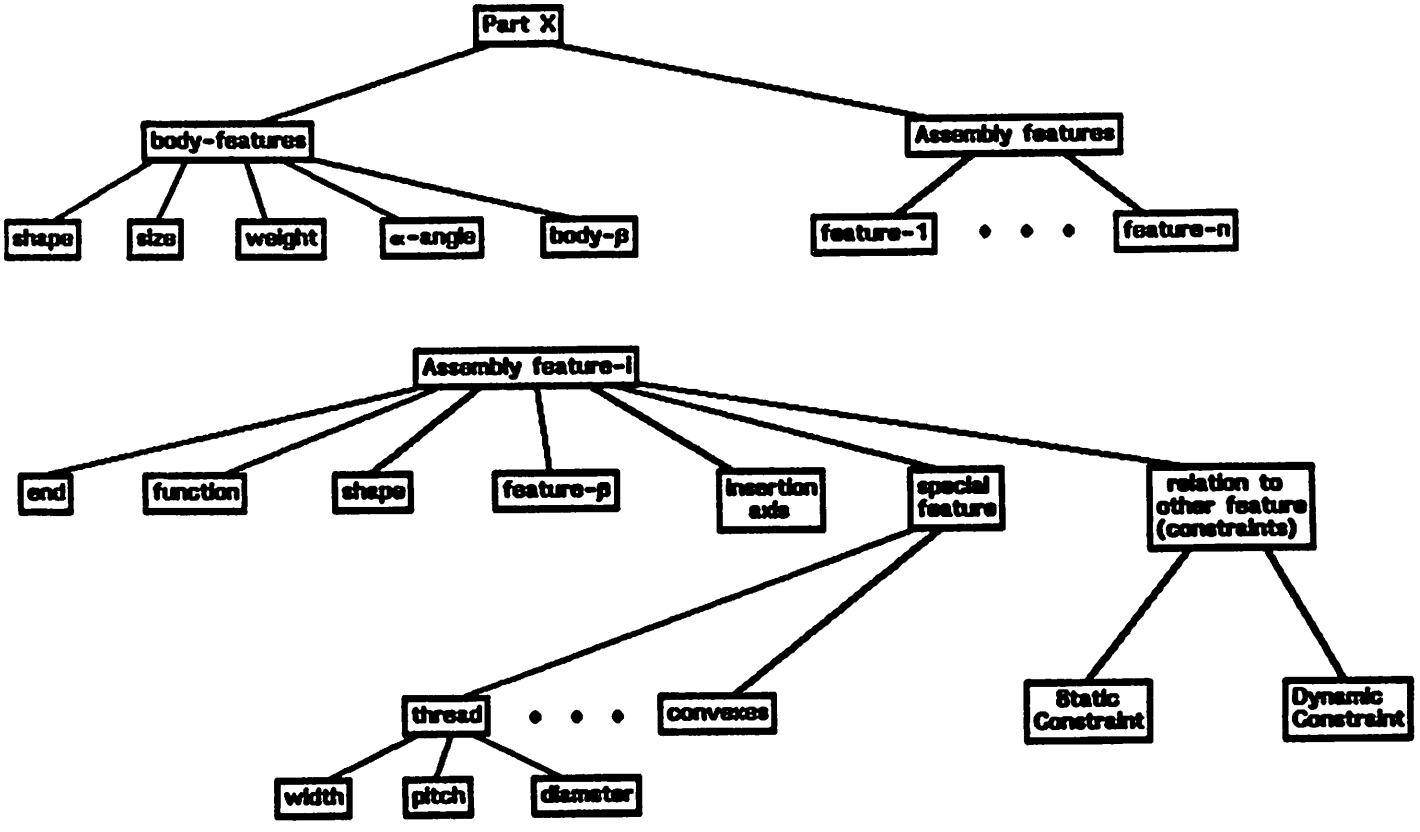


Figure 2: Assembly Object Hierarchy

2.2 Representation

2.2.1 Objects

There is a hierarchical representation tree for each physical object in the assembly task (as seen in Figure 2). Each object (part) is represented by body-features, such as shape, size, etc., and *assembly features* which may be defined as "a specific geometric configuration formed on the surface, edge, or corner of a workpiece intended to modify or aid in achieving a given function." [7]. Each *assembly feature*, or feature for short, can be further divided into even more detailed features (Figure 2) which are needed to specify more precise assembly actions. The parts used so far are all two-ended (along the central axis) parts, and the assembly features all reside at the ends. The *insertion axis of a feature* is the axis which is perpendicular to the feature surface and pointing away from the feature[3]. It is useful when trying to fit or thread two matched features together.

There are two types of roles a feature can play: either as an *insertor* or as a *container*. An *insertor* is defined as a convex shape of an assembly object which can possibly be put into a corresponding concave shape of another object which is called *container*.

In the implementation of the planner, the shape of each feature was represented as: round, half-round, rectangular, and for each combination of two features, say, fitting a half round shape insertor into a round shape container, the new "half round" feature that was born as a container would be represented explicitly in a rule. With the increasing varieties of shapes this symbolic representation approach brings lots of redundant implementation work. Each combination of two feature

[3]Note: CAD – Computer Aided Design – can provide the insertion axis of a part, which may or may not be the same as the insertion axis of a feature. Please refer to the next section for Dixon's example

shapes has to be represented explicitly. To make the match and the control of birth and death of features more general and easier, the following approach is proposed. Based on the idea that an insertor's shape can be represented as its circumscribed circle and container as its inscribed circle, and the matching of two features can be just simplified as comparing the two circles, polar coordinates seems to be a easier way to represent them. However, we also noticed that a container's inscribed circle not less than an insertor's circumscribed circle is just the sufficient condition for the two features match but not necessary. One example would be that the two features have the same shape (square, rectangular, etc.) but slightly different sizes (Figure 3. This left us the room to work on.

Figure 3

Feature's matching using inscribed circle and circumscribed circle

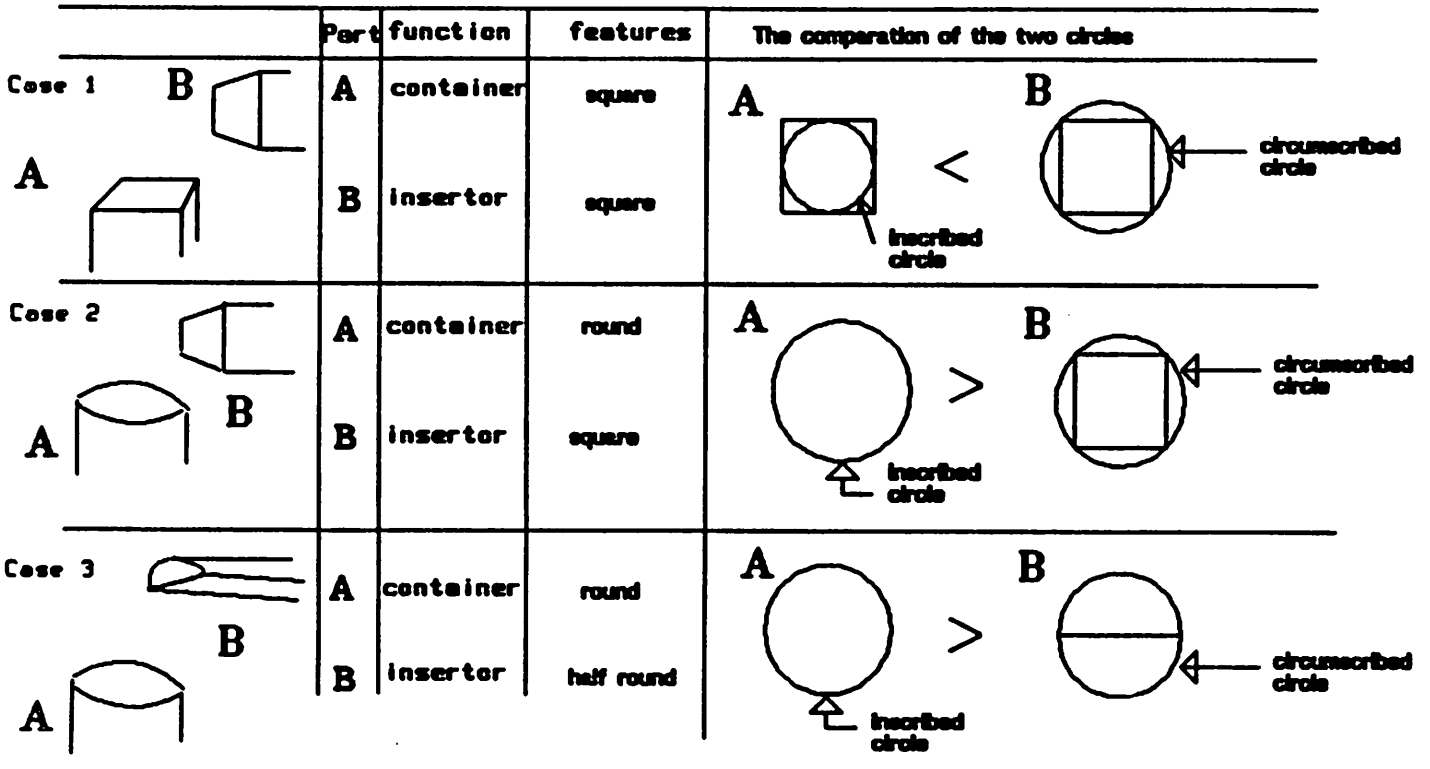


Figure 3: Feature's Matching using Inscribed circle and Circumscribed circle

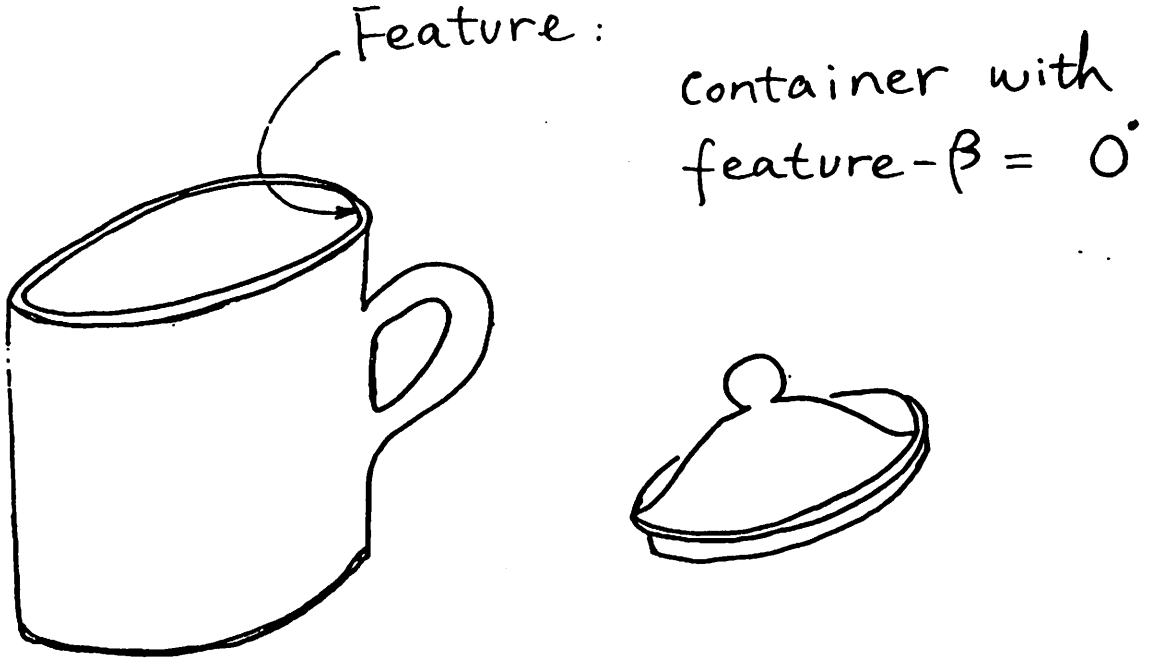


Figure 4: Difference between body-β and feature-β

During insertion, a part may have to turn around its insertion axis before it fits into the matched part. β-symmetry is the rotational symmetry of a part about its axis of insertion; or equivalently about an axis which is perpendicular to the surface on which the part is placed during assembly. The magnitude of rotational symmetry of the smallest angle through which the part can be rotated and repeat its orientation is the β angle of the part. For a cylinder inserted into a circular hole, $\beta = 0$; for a square section part inserted in a square hole $\beta = \pi/2$ etc.[5]. We have extended this concept from a measurement for the whole body of a part to a measurement just for a feature, calling it feature-β as opposed to body-β.

Similarly, α-symmetry is the rotational symmetry of a part about an axis perpendicular to the axis of insertion. For parts with one axis of insertion, end-to-end orientation is necessary when $\alpha = 2\pi$; otherwise $\alpha = \pi$ [5].

The distinction between body-β and feature-β is shown in Figure 4. The mug has body-β of 2π because of the handle but the container feature at one end of the mug has a central symmetrical feature-β, which means that it equals zero. The

feature-β value is used in a heuristic rule to guide the planner to look for more detailed commands.

If the planner finds that both matched feature-β values are zero then it can fit them together like putting a lid on the mug, i.e. if the matched feature has feature-β with value zero(central symmetrical) then no matter how much you turn the part around the insertion axis the result of assembly is unique. If one of the features has a feature-β more than zero, say, equal to π, then the number of ways to assemble it to another part depends on the feature-β value of the matched feature in the other part. If the planner finds a matched feature also with feature-β value π in another part, it knows that it must find more detailed information about the matched features in the hierarchical object representation shown in Figure 2. It would try to see if the detailed features which cause the π feature-β values match each other.

2.2.2 Constraints

Constraints are *the interactions between subproblems* [8]. The planner is concerned with both static and dynamic constraints. *Static constraints* are the interrelationships between features of the same part. *Dynamic constraints* are the interrelationships of features not residing on a single part, or goals which are formed and propagated while the planning is going on.

There are all kinds of relationships among the features of one object and the features of different assembly objects, such as: *central-related* which means that two or more features at one end of an assembly part have the same central insertion axes. For example, the two features of part A end two are *central-related* (Figure 7). One is an insertor with threads and the other is a container. The planner must check if part A and part C can be engaged with part B in between. If two features' insertion axes are not overlapped, as in Figure 8 the two features on part E are

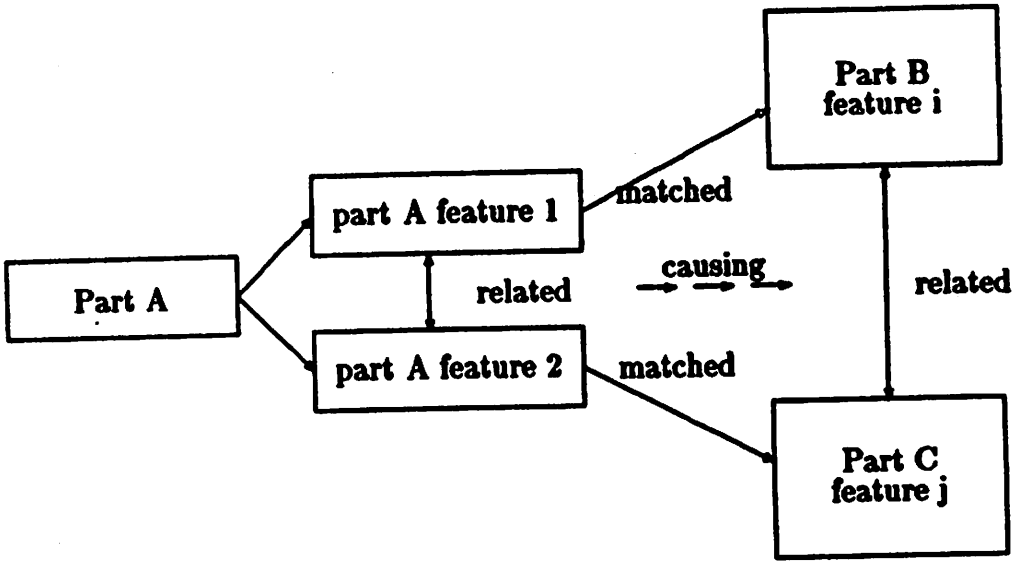


Figure 5: From relation to other features to relation to other parts

not *central-related*, even after part F is already fitted into one of the cylindrical holes of part E, the planner fits part G into another hole, totally ignoring part F. The explicit representation of the relations between features saves the planner from unnecessary work.

The feature relationships of a part – static constraints – can produce dynamic constraint(s), since they can be propagated during the procedure of assembly. For example, part A in Huber's first example has double features at end two and the two features' insertion axes are overlapped as seen in Figure 5. The planner can reason from this that part B and part C's insertion axes are overlapped too, since they are overlapped with part A's two overlapped features. This transitivity helps the planner to know the relative spatial positions of each part. This kind of knowledge can help the planner to decide which end of part D should go with part B, it knows that part C is outside part B, and that part D's intruding edge end should go into the space between part C and part B. This information can also help the planner reason when it checks if the two matched features can be physically engaged. If

a matched feature has a central-related feature, the planner must check out if the central-related feature has been fitted or threaded to other features in order to make sure the two matched features can be physically engaged. If neither of the matched features has central-related features, the planner can be sure without checking that the two matched feature can be physically engaged.

2.2.3 Operators

We represent operators in the form of if-then rules. Their preconditions are represented explicitly in the IF part of the rule, and add-delete lists(the list of the facts added and deleted after the execution of the operator) will be represented implicitly. For more detailed description, please see section 2.3 Rule-base.

2.2.4 Goals

The planner is just an intermediate link in a complete intelligent assembly system that our lab is developing with high level assembly commands as the input and low level action commands to a robot hand as output.

The input to this planner is one or more assembly instructions. These input instructions are *goals* to the planning system, each one of the goals is in the format :

Operation Object1 to Object2

such as:

Fit part B to part A (as seen in Figure 4)

The output of the planner is one or more assembly commands(for each goal) which refine the input goal. An example would be :

fit part B end1 feature1 to part A end2 feature1 (as seen in Figure 4)

This format of goals has some extra information: *object1* is an *insertor*, and *object2* is a *container*. Associated with each goal are two preconditions to be instantiated while the planning is going on:

a. there is at least one pair of matched features in the mentioned parts;

b. the two matched features can be physically engaged.

Each goal also has an add-delete list – a list which describes what facts were added and/or deleted after the goal **plus** the previous goals was reached. The add-delete list is not represented explicitly since it depends on what the previous goals are.

2.2.5 Search space

We refer to all the possible choices the planner can make as its *search space.* It is represented as a search tree; one choice is represented as one node on the tree. Using the inference mechanisms built into ART, the planner would put every heuristic rule that is triggered into a list and randomly pick one to fire. It would continue this process in a depth first search manner, i.e. if the result of the fired rule caused one or more rules to trigger, then one of these would fire and so on. If no new rules triggered then the planner would backtrack to the place where it made the latest choice, and take another branch.

Effort was paid to minimize the search space as well as the backtracking.

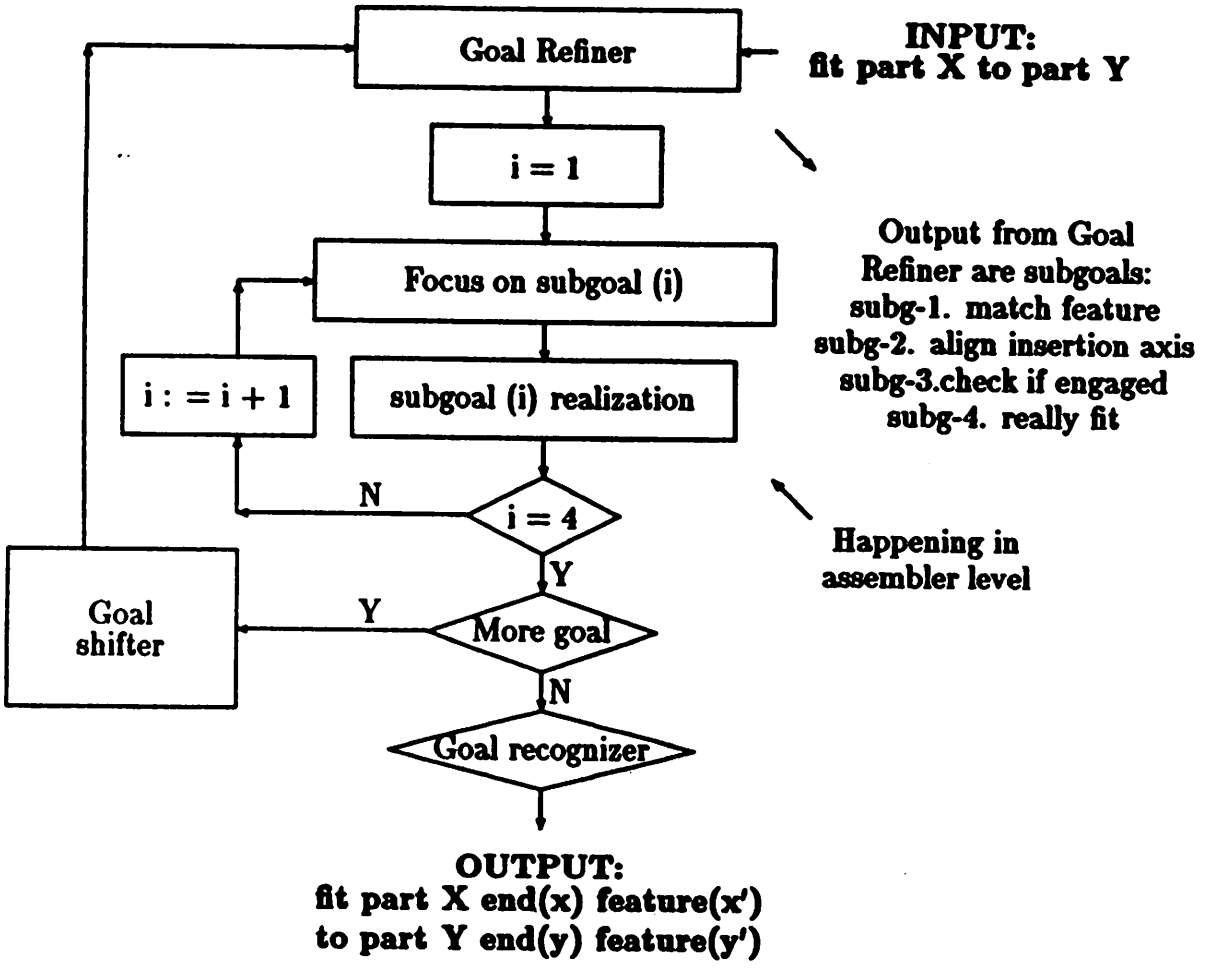


Figure 6: Planner Executing Loop

2.3 Rule-base

The planner can currently handle two kinds of operations: FIT and THREAD.

The input to the planner is a set of unordered goals. First, the goal refiner at the planner level refines the focused goal to four subgoals (Figure 6) and puts the subgoal pointer on the first subgoal *match features*. Second, the assembly level operators are invoked to realize the current subgoal. After one subgoal is finished the pointer moves to the next subgoal until four of them are all reached. The planner then checks to see if there is any more goals left. If there is more goal, the goal shifter at the planner level shifts the pointer from the current finished goal to the next goal and the above actions repeat; otherwise goal recognizer is invoked to claim either the task is finished or failed. The output of the planner is an ordered set of commands describing how to put the parts together in detail (Figure 6). All the goals are reached in a non-conflicting order.

The planner examines the relevant pre-stored information, including both static and dynamic knowledge (Figure 1). It uses heuristics at each level to figure out which ends and which features of the two parts as mentioned in the goal should go together. The planner has three main parts in static knowledge base (LTM) (Figure 1): the knowledge base containing the object hierarchy, the rule-base containing heuristics and the goal-base developed from the input goals. The rule-base is composed of heuristic rules which can be further divided into three kinds: assembly heuristics which work on real physical parts at assembler level, goal order heuristic rules which work on goals and subgoals at the planner level; and meta-planning rules for controlling the attention of the planner, guiding the planner among the planning levels[4]. The problem-solving state data are kept in a global dynamic knowledge base called blackboard(STM) (Figure 1). The dynamic constraints, possible matches of the features, and cross-level control information generated by each heuristic during

[4]See Table 1

planning are stored in the blackboard.

Followings are rules used in the planner. "IF" part of the rule is the preconditions for the rule to fire. If all the preconditions (either from LTM or STM) are satisfied, the rule fires and *posts* the relevant information on the blackboard. The information updates STM. Traces for each example are at the end of this report including all the information posted by the planner.

Heuristic rules for assembly:

These are operators used in assembler level.

1. Feature match for *thread* operation:

IF
the focused goal is "thread X to Y";
AND X has a feature(x) at end(x') which is an *insertor*
AND Y has a feature(y) at end(y') which is a *container*;
AND both features have threads;
AND the sizes(diameters, pitches, widths) of the two features are compatible;
THEN
post "feature-matched part X end(x') feature(x) to part Y end(y') feature(y)".
(*post* here means to post on the *blackboard*)

2. Feature match for *fit* operation:

IF
the focused goal is "fit X to Y"
AND X has one feature(x) at end(x') which is an *insertor* and Y has one feature(y) at end(y') which is a *container*;
AND the size of feature(x) is smaller than the size of feature(y)'s
THEN
post "feature-matched part X end(x') feature(x) to part Y end(y') feature(y)".

3. Align insertion axes (this is a constraint)

IF

"feature-matched part X end(x') feature(x) to part Y end(y') feature(y)";

AND the insertion axes of feature(x) and feature(y) are not aligned along the same axes;

THEN

post "align-axes part X end(x') feature(x) to part Y end(y') feature(y)" Since this planner does not care about collision-free trajectories, it does not mention how to move the parts to merge the axes at all.

4.Check if the matched features can be engaged

After two features are matched to each other, this rule is invoked to see if the two matched features can be physically engaged. There are several possibilities:

1. The simplest case is that both features have no other central-related (refer to 2.2.2. Constraints) features then no possible obstacle in the way;

2. If one of the features has central-related feature(s), but no other feature either fits or threads into it, then the two matched features can be engaged;

3. If a feature central-related to one of the matched features has been fitted or threaded with someone else, then it is needed to invoke the sensor to check out if the two matched features can be physically engaged.

IF

"feature-matched part X end(x') feature(x) to part Y end(y') feature(y)";

AND there are no central-related features to the matched features

THEN

post "feature-engaged part X end(x') feature(x) to part Y end(y') feature(y)";

ELSE (there is a central-related feature)

IF

nothing in between

THEN

post "feature-engaged part X end(x') feature(x) to

part Y end(y') feature(y)";
ELSE(there is a central-related feature and something joined to it)
IF
there is something already ***joined*** to the central-related feature which is at the same end of the matched feature
THEN
invoke the sensor to see if the two matched features can be engaged.

5. Fit together

IF
"feature-engaged part X end(x') feature(x) to part Y end(y') feature(y)";
AND at least one of the matched features' β values is zero (the feature is central symmetrical);
THEN
post "no matter how much you turn around the insertion axis to fit part X end(x') feature(x) to part Y end(y') feature(y), there is a unique assembly result"
ELSE (none of the features' β values equals to zero)
IF
both features' β values are equal
THEN
post "check lower level features, possibly only one way to assemble part X end(x') feature(x) to part Y end(y') feature (y), and no more than β degrees needed to turn around insertion axes to fit them together";
ELSE(features' β values are not equal)
post "more information needed to decide how to assemble them together".

Heuristic rules for goal (subgoal):

1. Goal refiner:

This rule refines a given goal to ***four*** subgoals (Figure 6) and locates the subgoal-focusing-pointer on the first one – match features.

IF
goal X is currently the one which is focused on
THEN
Post "Subgoals for goal X are:
1. match features,
2. align axes,
3. check if the matched features can be engaged,
4. finish the goal by doing the operation mentioned in the goal;
focusing on the first subgoal – match features".

2. Subgoal shifter

IF
subgoal(i) is done
AND i is not equal to four
THEN
Post "focusing on subgoal(i + 1)".

3. Goal shifter

IF
subgoal(4) is done
AND more goal(s) is(are) following;
THEN
Post "focusing on the next goal".

4. Goal order:

When the input to the planner is a set of unordered goals, this rule fires to order the given goals. This heuristic is based on the way how human beings handle blocks.

IF

goal X: (operation1 object1 to object2)
AND goal Y: (operation2 object2 to object3);
THEN
Post "goal X should be ordered after goal Y unless Goal order heuristic two (next) is violated".

5. Goal order:

IF
the success of goal X deletes the precondition of goal Y
THEN
Post "goal Y must be ordered before goal X".

Meta-planning rules

1. Digging more information for the decision of goals order[5]

IF
the order of the goals cannot be decided by heuristic rule one for goal ordering above,
THEN
Post "work on the assembler level to find out the add-delete lists for each goal".

2. Look up the following related goal's content at the planner level for the decision in the assembly level[6]

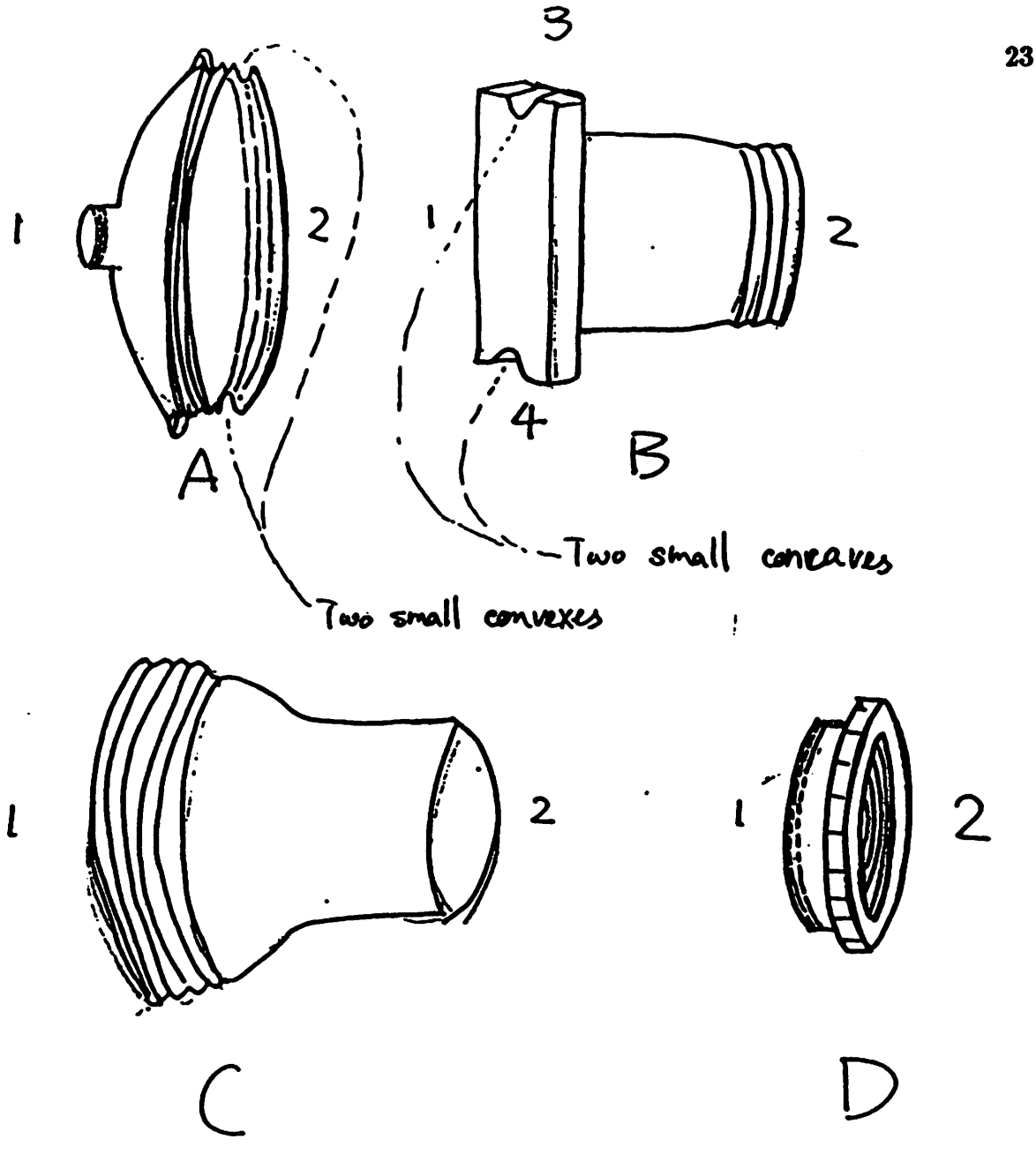
IF
the way to assemble two matched features cannot be decided
THEN

[5]Please refer to section 3.5, Rukmini's example, for a detailed explanation

[6]Please refer to the Discussion Section – Control part: the combination of control strategy and representation hierarchy

Post "Look up the content of the following related goals
in the planner level".

The goal-base is a set of predefined final states of the parts features' relations. As soon as the relationships established by the assembler level operators all fall into the final state set, a rule called *the goal recognizer* will fire and claim that the goal is reached.



Goal: A2 — B1 fit

A2 — C1 thread

B2 — D1 thread

Figure 7: Huber's First Example

3. The history of the planner

The planner developed from working in a single-example[7] model to working in a multi-example model. It currently works on five examples. As assembly examples were added in, the planner expanded and changed in the representation of the parts, the heuristics and the planning control. The growth and development of the planner will be shown by describing its behavior in terms of the examples. Appendix A containing computer-generated traces is at the end of this report.

3.1 Version one – Huber's first example

In the initial stage, the planner was designed as a forward reasoning system [9] involving heuristic and constraint rules, aimed at solving Huber's first example, as shown in Figure 7. The only constraint it had was:

Always keep the parts central symmetric about their central symmetric axis.

It contained the following heuristics:

1. If the goal mentions that fit part A and part B then look for corresponding special features on the two parts, such as two convexes in A and two concaves in B, then fit them together.

2. Look for matching threads, i.e. if both parts have threads, one is inside and one is outside, and the diameters are nearly the same.

3. If the threads are matched then check to see if they can be physically engaged by invoking the sensor (actually in the simulation state, it asks the user for this sensory information)

4. If the threads are matched and can be engaged then thread them together.

All parts in the example are central symmetrical except one end of part B which is rectangular. The planner was given a set of goals:

[7]Huber's first example Figure 7

1. Fit part B to part A
2. Thread part A to part C
3. Thread part B to part D

The output of this version was a set of detailed commands:

1. fit part B end one to part A end two,
2. thread part A end two to part C end one,
3. thread part D end one to part B end two.

The result indicates that the planner figured out which end should go with which. It didn't worry about the order of the goals. When there was more than one possibility for matching two parts, the planner could wait while triggered rules fired one at a time. As soon as the predefined final state was reached, it would claim the goal was reached.

This version was very specific to Huber's first example in the sense that it only worked on central symmetrical parts. One thing learned was that backtracking should be avoided as much as possible since it is very expensive in practice. In implementation you can see a huge search tree on the screen.

3.2 Version two – Huber's first example

Since the first version seemed too blind in the rule firing, the next stage was to develop a backward chaining [9] version. The goals were presented one at a time to the planner which refined the given goal into subgoals which were reached in the order noted in Figure 6. After one goal was completed, the next goal was given. When the planner reached the predefined final state, the goal recognizer claimed that the goal state was reached.

Clear distinctions between the planning levels were made in this version, as noted in Table 1. Although this version was much more goal directed than the previous one , it was overly restrictive in that the order of realizing each subgoal

was static even though this was not necessary. This version also worked only on central symmetrical parts, and still had to backtrack if a match was not the one which was wanted in the detailed goals descriptions.

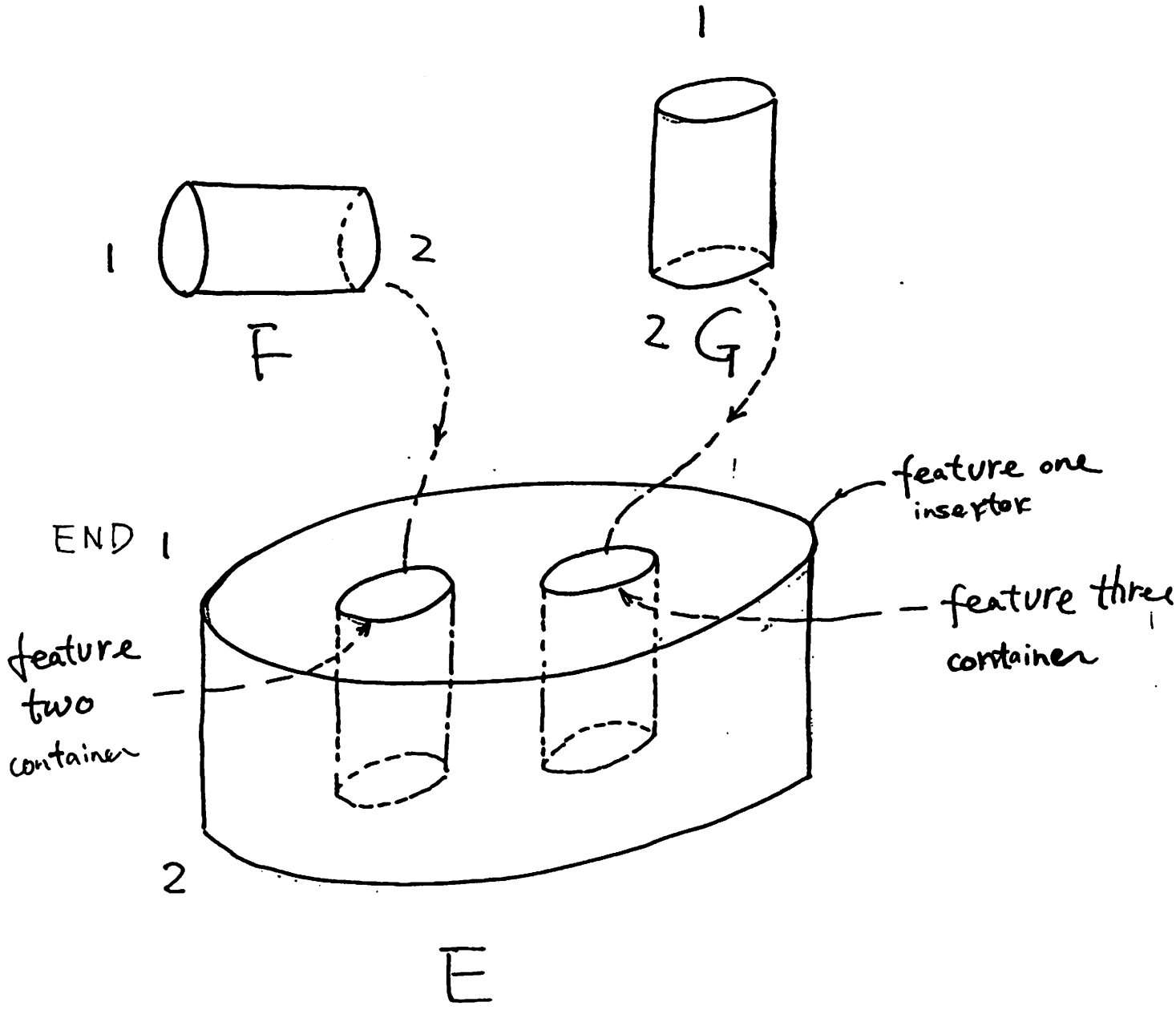


Figure 8: Arbib's example

3.3 Version three – Arbib's first example

Arbib's first example (see Figure 8) consists of parts that are not all central symmetrical, such as part E. As a result, the central symmetry constraint to keep all the parts central symmetrical all the time, and the alignment heuristic rule to align the two matched parts' central axes are no longer valid. In addition, one end may have more than one assembly feature. For example, part E has three features at one end: one is an insertor with an axis through the middle of the big circle, and the other two are containers with axes through the centers of the small circles. In this version, taking a cue from [10], the representation for each part changed from representing one end as a whole to representing the features at each end separately. More attention was paid to the order in which the align and match subgoals are reached, since only after the features match does the planner know where to align the corresponding axes. For example, part F should be aligned to the center of one small circle of part E and part G should be aligned to another center.

Two major improvements of representation were: the change from representing the end as a whole to representing each feature separately, and the change from the central axes of a part to the insertion axes of a feature. This established the foundation of the final representation hierarchy(as seen in Figure 2). The knowledge about the parts is much more local, functional and relevant to the assembly task. This made the planner more powerful by allowing it to recognize the relevant assembly features when handling multi-featured parts. Please refer to trace case 2 for a detailed description of the whole procedure.

	feature	shape	size (diameter)
part H	container	round	1.0 cm
part I	insertor	halfround	1.0 cm
part J	insertor	halfround	1.0 cm

feature: container

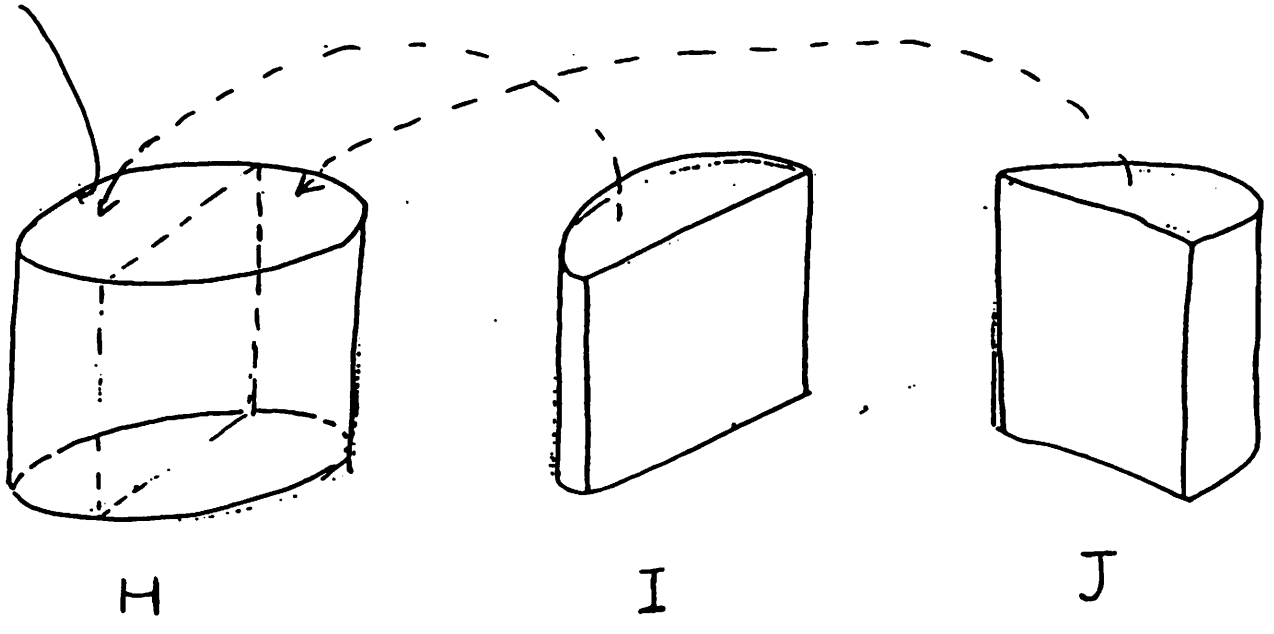


Figure 9: Yanxi's first example

Feature's birth and death

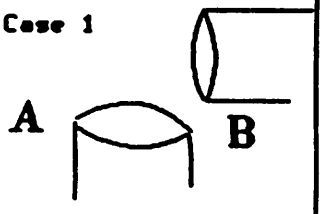
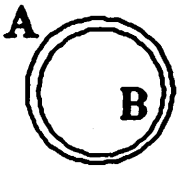
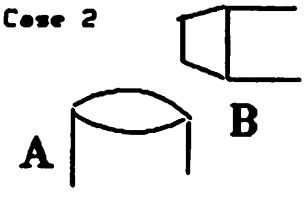
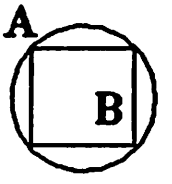
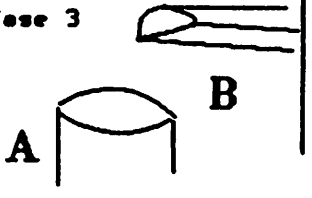
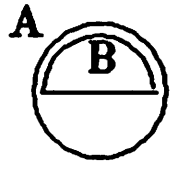
	Before assembly			After assembly	
	Part	function	features		features
Case 1 	A	container	round		Old features are disappeared, no new one is born.
	B	insertor	round		
Case 2 	A	container	round		Old features of both A and B are disappeared. Four new features of A are born. They are containers with  shape.
	B	insertor	square		
Case 3 	A	container	round		Old features of A and B are disappeared. One new feature of A is born. It is a container with half round shape.
	B	insertor	half round		

Figure 10: Feature's birth and death

3.4 Version four – Yanxi's first example

As shown in Figure 9, Yanxi's example yields a new problem: a feature of one part (part H), needs to match and fit with features of two other parts – part I and part J. The way the planner deals with this is to hide the feature, in the sense of making it not available anymore. New features are born, and explicitly represented after assembling two parts together. The birth and death of the features is dynamic and constrained by the shape and size of the two matched features. As shown in case 1 of Figure 10, there are times when no new features are born after assembly but the old features die; in case 2 not only do the old features die but also new ones are born; in case 3 an old feature dies and new ones are born. This version of the planner contained basic shape-combinations to let it know when and which feature(s) should be hidden or created. The planner handled the example in Figure 9 successfully by hiding the two old features of part H and part I and creating a new feature for part H after fitting part I into part H. This new feature had the function of a container and a half round shape. Part J could then fit into the other half of part H smoothly (Please refer to Appendix A case 3). The proposed new way to represent the shape of the feature as a function of polar coordinates as described in the representation and discussion section could be used, which would make the matching and the birth and death of features more general.

One beneficial *side effect* when using this version on the second example(Arbib's example) is that the search space is reduced – the goal state is reached at stage 19 instead of stage 27. The reason is that when a feature is matched, it is hidden along this branch of the whole backward chaining tree, so no more matches can be made (Please refer to Appendix A case 2).

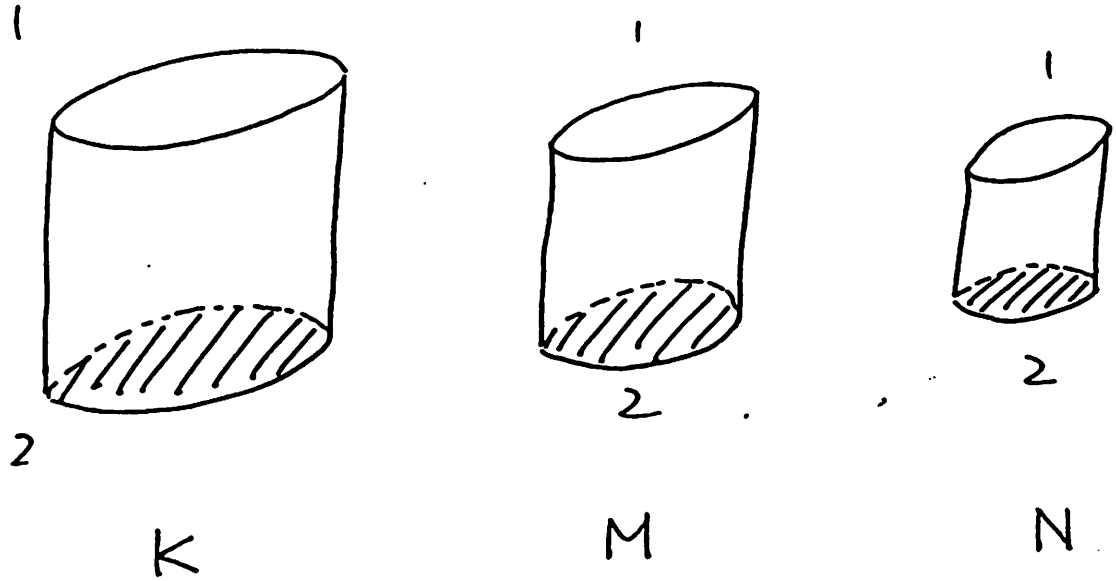


Figure 11: Rukmini's Example : Fit N to M and fit M to K

3.5 Version five – Rukmini's example

Rukmini's example is shown in Figure 11. Each part has one end closed and one end open. Goal ordering at the planner level was built into this version. A set of goals is given together instead of one at a time, and two heuristics were added to order them at the planner level:

1. **IF**
   There are goals :
   "operation-x A to B"
   AND "operation-y B to C"
   THEN
   order goal "operation-y B to C" first, except if this order violates the second heuristic(next)

   This rule is suggested by the way human beings do the assembly.

2. **IF**

the total delete list of goal X **plus** the result of completing the goals before goal X, would delete the precondition of goal Y

THEN

do goal Y first

The second rule requires either that the add-delete lists of each goal be represented explicitly, or that the lower assembler level be invoked to find out the implicit delete list of each goal.

At the planner level, the goals and constraints become objects for the planner to work with. Generally, the preconditions for each goal are:

a. there is at least one pair of matched features in the corresponding parts;

b. the two matched features can be physically engaged.

Rukmini's example is solved by the planner using the first heuristic rule in the planner level. It saves the planner from looking for more information which is not required for the decision about the order of the goals. Please refer to the trace in Appendix A case 4.

Using this version on Huber's first example (Figure 7), some problems occur. The input goal set is:

Goal X — thread part B to part D

Goal Y — fit part B to part A

Goal Z — thread part A to part C

The planner decides that Goal Y has to come before Goal Z according to the first heuristic rule, since Goal Z is in the way of accomplishing Goal Y. It still does not know where to put Goal X. There are three choices: before, in between, and after Goal Y and Z. At this time the planner invokes the lower level – to find out the matched features and see if this can provide any information to help the planner decide the order of goals. This gives the planner the flavor of least commitment strategy (delay making the decision as late as possible until there is a reason to do

so).

One of the preconditions for goal Z is: the two matched features, from part A and part C respectively, can be physically engaged. Performance of the goal X and goal Y deletes this precondition[8]. So goal X and goal Y cannot both be ordered before goal Z, and goal Y's position is already decided, by the first heuristic, to be before goal Z. The only position left for goal X is behind goal Z. The planner gets the final order of the goals as:

first — goal Y: fit part B to part A;

second — goal Z: thread part A to part C (by the first heuristic rule);

third — goal X: thread part D to part B (by the second heuristic rule).

How can the planner find out goal X deletes the precondition of goal Z? There are at least two ways: A dynamic planner can determine it by invoking the use of a sensor, which is simulated by asking the user. By calculating the necessary information in its database, say the widest size of the part and the shape of the part body, the planner can determine whether one goal reached can be in the way of another goal. There is a tradeoff between using the sensor at run time and retrieving off-line knowledge from a database. Using a sensor is more expensive than data retrieving; however, due to the dynamic nature of the environment, the static information may no longer be true.

---

[8]Note: this is an example of dynamic constraints which do not exist before the assembly. Also note: only the sum result of goals X and Y deletes the precondition of goal Z, neither X nor Y does that by itself.

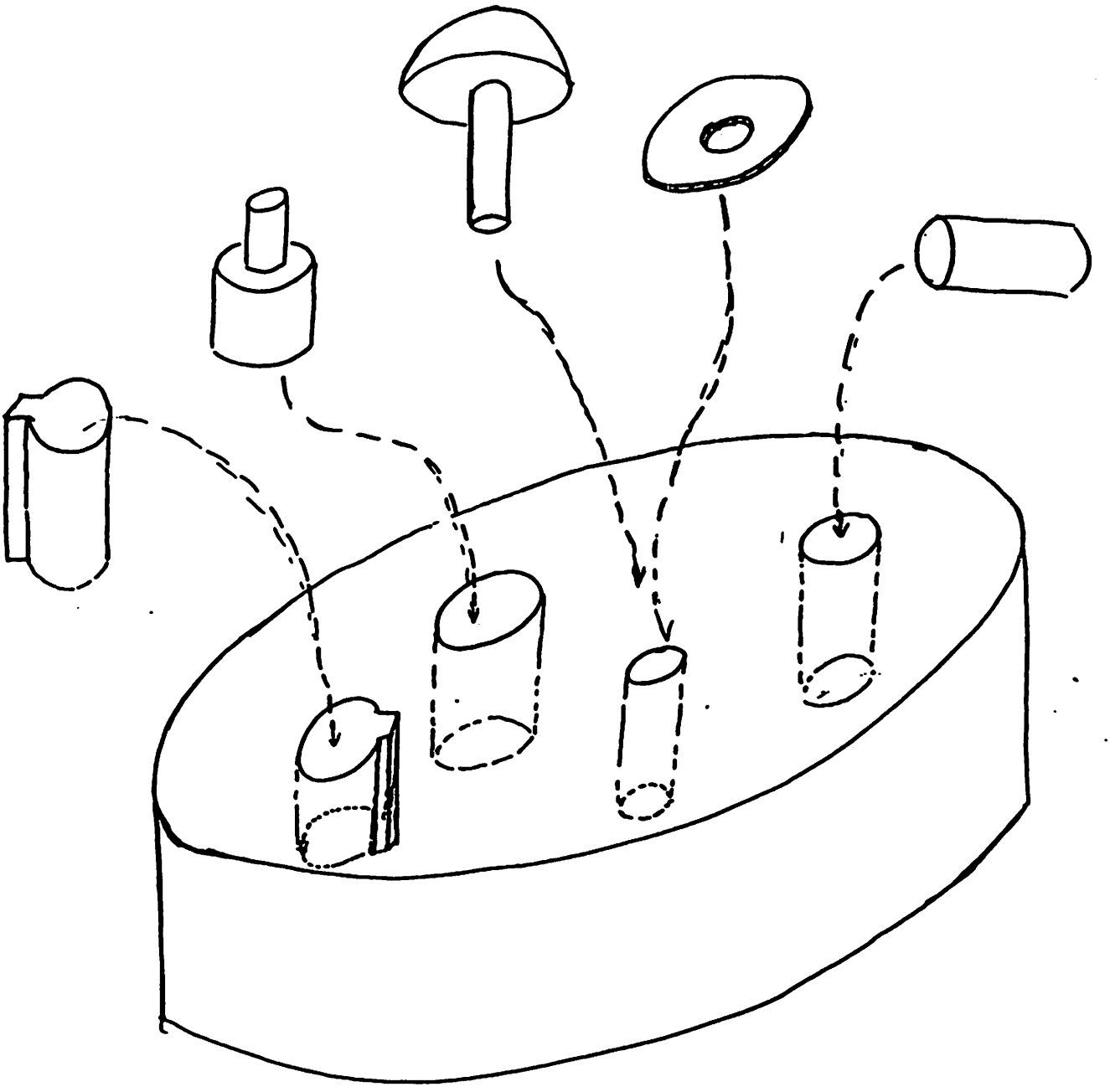


Figure 12: Dixon's example

3.6 Final version – Dixon's example

Dixon's example is shown in Figure 12. This version of the planner can recognize detailed goals by using β and α angles (as outlined in section 2.2.1), and give detailed commands for realizing the detailed goals.

After adding α and β values of the parts to the object hierarchical representation, two new heuristics are added:

1. **IF**
   At least one of the matched features has feature-β value equal zero,
   THEN
   No more search is needed to decide how to assemble them together
2. **IF**
   Both matched features β values are equal
   THEN
   Check to see if the corresponding detailed features match.

Using these heuristics in Huber's example (Figure 7), the planner finds out that part A's two convexes and part B's two concaves can fit together successfully and gives more detailed commands: *fit part A end two feature two two-convexes into part B end one feature one two-concaves* (traces case 1).

When a part has an α-angle equal to π (both ends of the part are the same),the planner knows that it does not matter which end to fit or thread to a matched feature in another part since it is π α-symmetrical. For instance, in the second example (Figure 8), although both ends of part F match part E's one feature, the planner does not care which end of part F fits into part E since part F's α-angle is π(the two ends are identical). However, while executing a real assembly task, the two identical ends of a part do make a difference since, for instance, one end can be physically nearer to the matched feature of another part than the other end. In this case, the output of this planner contains information saying that there is no

difference to fit/thread this feature at this end or the other identical end, leaving the decision for the next lower level planner. This helps the planner prune out matches which are the same before doing the matches, instead of choosing one after all the redundant matches are already done.

In addition, an α angle can help the planner find more useful information when it needs to. In Huber's first example, both ends of part D can match part B end two, but the planner notices part D's α-angle is 2π and it tries to find more information to decide which end of part D should go with part B before it makes the guess or randomly picks one. A similar thing happens in Rukmini's example. Part M with α equal to 2π has two features which look the same, but are at different ends, and which match with part K. To decide which one is supposed to be the right one the planner looks up the content of next goal: "fit part N to part M"; since part M has one end open and one end closed, the planner picks up the feature which guarantees the accomplishment of the next goal.

Two more heuristics are added:

3. **IF**
   One of the matched features has α equal to π
   THEN
   The number of the matches can be reduced in half

More generally,

3. **IF**
   One of the features, called feature(a), on a part matches to another feature, called feature(b), in another part
   AND the first part has body α value which is a divider of 2π
   THEN
   there would be $2\pi/\alpha$ identical matches among the features in the first part and feature(b) in the second part, only one of them need be followed.

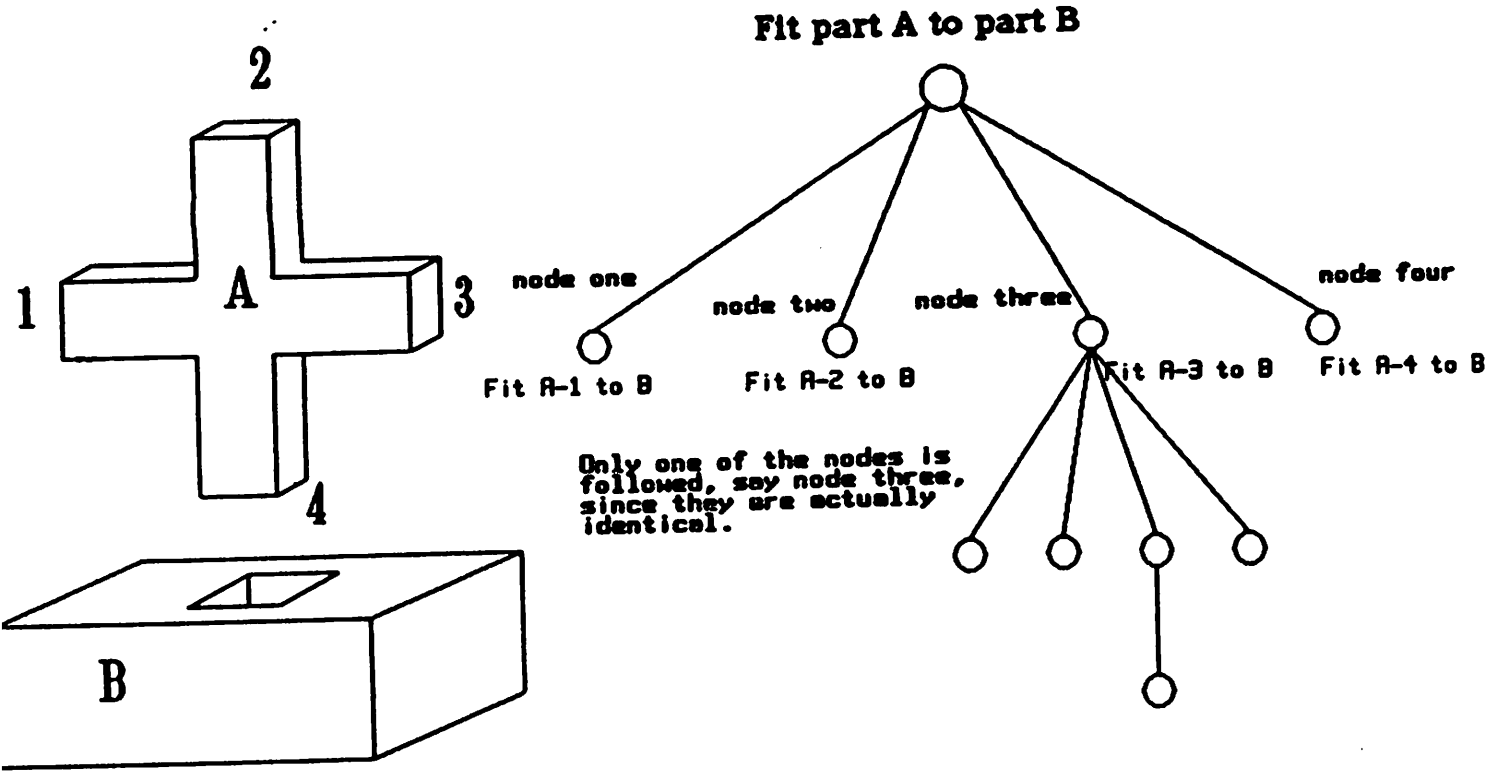


Figure 13: An example of multiple choices search tree

For example, in Figure 13 part A has $\alpha = \pi/2$. There are four $(2\pi/\ \pi/2)$ possible matches between part A and part B feature 1. The planner only follows the match by part A feature 1 and part B feature 1 and ignores the others.

4. **IF**

Part(x)'s two ends both have a feature(s) matched to another part, *AND* part(x)'s α value is 2π

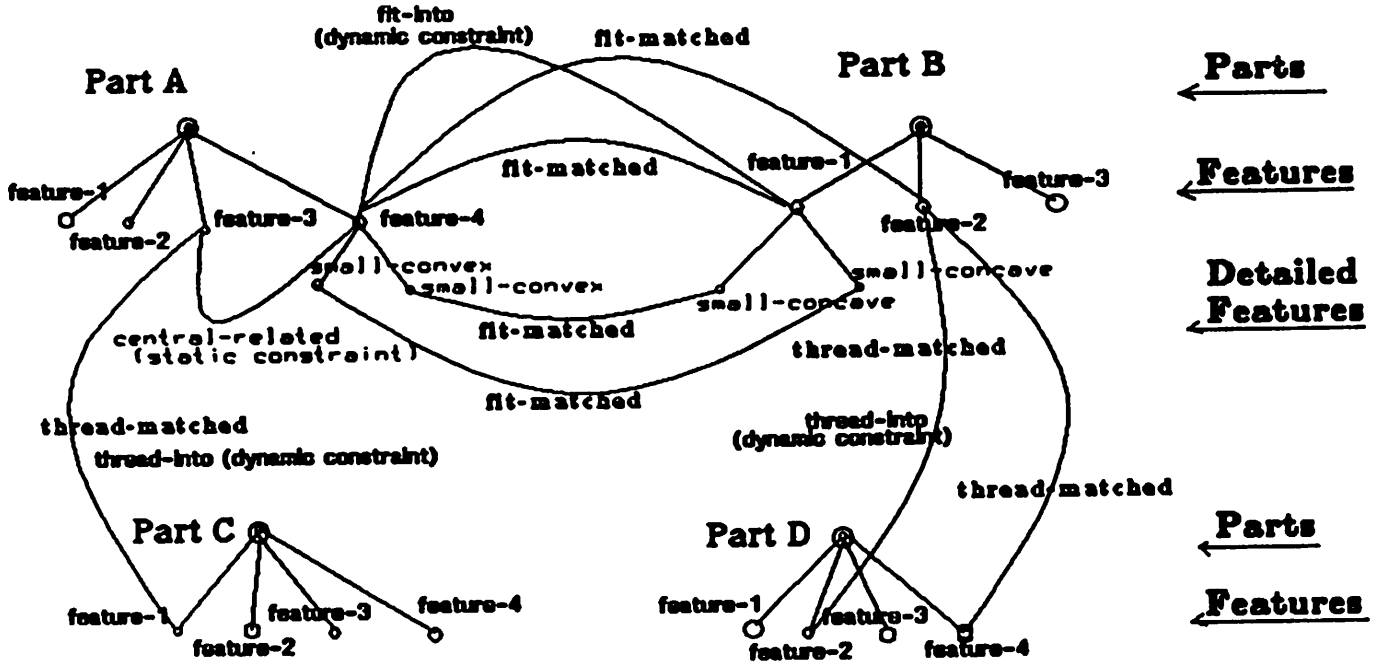
THEN

to decide which feature of part(x) should be really matched, look for more information. Either reasoning from partially assembled result or look for the content in the following goal(s) if there is any.

As mentioned above, the planner looks up the content of the next related goal(s) in Rukmini's example when trying to assembly M to K ; and examines carefully the spatial relationship of the partially assembled parts in Huber's example when fitting part D to part B.

Both β-angle and α-angle function like flags to guide the planner to make wise moves, pick up a node which is worthy to develop further. and to avoid it when it is redundant. This helps the planner to do a guided search instead of a totally blind search and reduces backtracking to a minimum.

Figure 14 A Layered Representation Graph For Huber's Example



	Part A				Part B			Part C				Part D			
Features	1	2	3	4	1	2	3	1	2	3	4	1	2	3	4
End	1	1	2	2	1	2	2	1	1	2	2	1	1	2	2
Function	cont.	inst.	inst.	cont.	inst.	inst.	cont.	cont.	inst.	inst.	cont	inst	cont	inst	cont
Shape	round	round	round	round	rectan-gular	round	round	round	round	round	round	round	round	round	round
Thread	yes	no	yes	no	no	yes	no	yes	yes	no	no	no	yes	no	yes

Features for Huber's first example

(cont. = container inst. = insertor)

4. Discussion

In this section we analyze the representation and control used as the planner developed and discuss potential room for improvement.

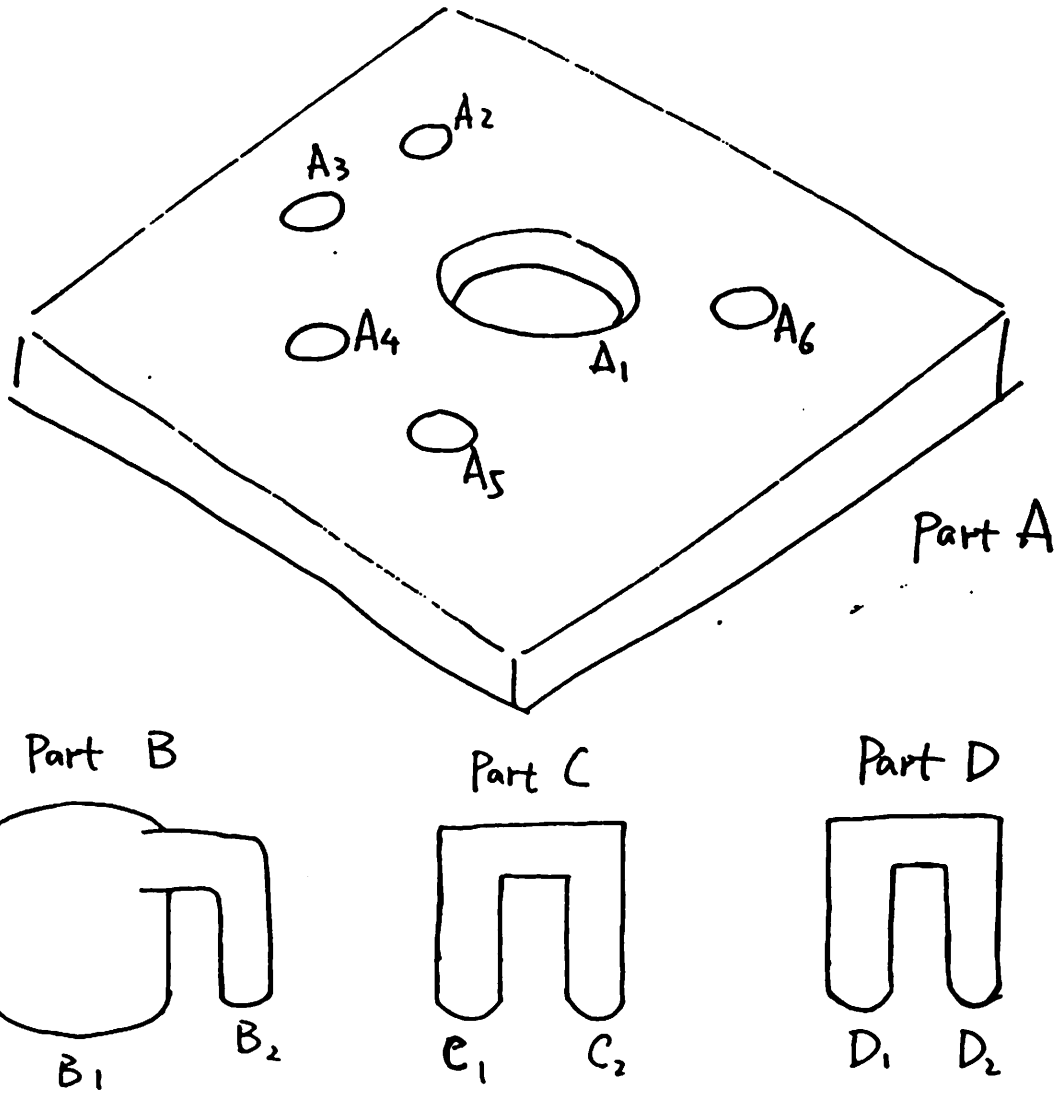
4.1 Hierarchical Representation

As was shown in Figure 2, the representation for each assembly part is a tree. Parts are represented by their features, and each feature can be divided into even more detailed features which are needed for more precise assembly commands. Maintaining an assembly feature's relation to other features is very useful, in that this is the place where the static constraints reside and the place where dynamic constraints are attached. (2.2.2) The introduction of constraints turns the hierarchical representation tree into a layered representation graph. It seems that if what the planner does is actually establishing relationships among the features of different parts, these relationships could form a semantic network [9] with the assembly parts and features as nodes, and relationships to other parts and their features as the arcs. This would lead us to examine the assembly task from a more general and abstract perspective.

Figure 14 shows a layered representation graph for Huber's example (Figure 7). The graph has three layers. The top layer is parts, second layer is features and the bottom one is detailed features. Part A and B have detailed features but part C and D do not. Some relationships are established at each level, such as, central-related, thread-matched and fit-matched in the features layer; fit-matched in the detailed features layer. Central-related is a kind of static constraint and fit-into is a kind of dynamic constraint (2.2.2 Constraints). This figure shows the result after assembly.

For the example in Figure 15, we could either represent the two features at one end as one feature or use the "relation to other features" slot in each feature as

a pointer to other tightly related feature(s). The former representation is more straightforward but we prefer the latter one since that it is more flexible and has more potential ability to handle complicated cases. The concept of constraints is especially useful when the planner deals with the example in Figure 16, Corkill's example. The relative spatial positions between the parts (relations of the features and the parts) are very important for the planner to decide when the final goal is reached, since the given goals can lead to ambiguous results.

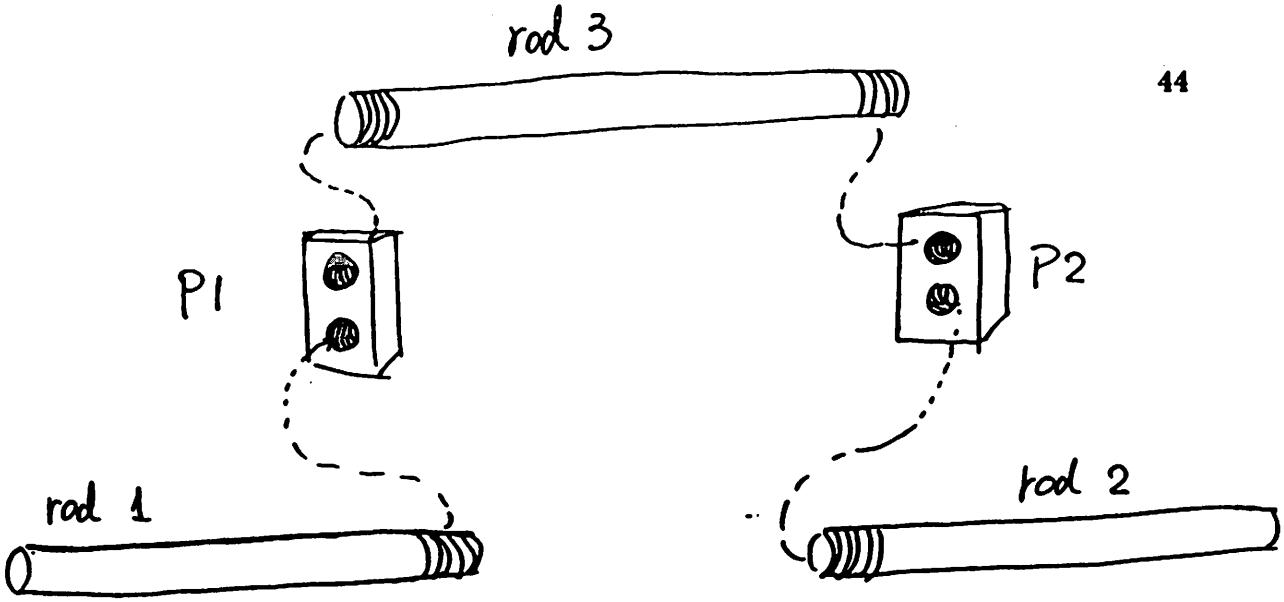


Insert B in A

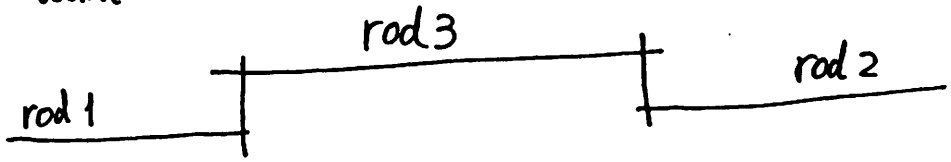
Insert C in A

Insert D in A

Figure 15: Torras's Example



result wanted:



Thread	rod1	to	P1
Thread	rod2	to	P2
Thread	rod3	to	P1
Thread	rod3	to	P2

Figure 16: Corkill's Example

4.2 A simple representation for features

In addition to find out if two features are matched, the planner also needs to decide what is left after fitting two features together, that is what we called the birth and death of the matched features. The representation for features plays a very important role in these activities. As we mentioned in 2.2.1 Object Representation, we are trying to avoid the redundant work caused by using symbolic representation for each feature shape. We are also trying to find out a better way to handle the birth and death of matched features. The following method is proposed but not necessarily the best.

Let us start with a two dimensional space. The origin of the coordinate frame of each feature is located at the center of its circumscribed circle if it is an insertor or inscribed circle if it is a container. A shape function for each feature is introduced:

The outline of a feature can be represented as a function under polar coordinates, F(α, r). The match and the birth and death of the feature can then be decided by the difference of the two(insertor and container) feature's shape functions with the two origins of the coordinates overlapped. If the difference is greater than or equal to zero then they are matched. If everywhere the difference is equal to zero or less than a certain value then old features die and no new feature is created as in case one in Figure 10. Otherwise, one or more new features are going to be born! If there are N continuous parts of the difference equal to zero or less than a small value, then there are N new features born as in case two in Figure 10. If there is one or no continuous part with a difference less than the small value then there is exactly one new feature born as in case three in Figure 10. The shape function F(α, r) can be expanded to a three dimensional function by considering the length of the feature along the insertion axis.

There are several questions to be discussed:

1. where is the axis supposed to be locate at the feature surface if the shape is

not central symmetrical?

2. which inscribed circle should be chosen if a container has more than one of them, say a rectangular shape container(actually infinite inscribed circles)?

3. how is the new born feature's shape function created efficiently?

Some of our near future work involves finding solutions to these questions.

4.3 Control among the levels

The structure of the system(Figure 1) follows MOLGEN's hierarchical planning idea, with levels of control. For all of the examples, the planner stays in a least commitment mode which is one of the supervisor level modes. The planner can work in one of two versions, either with a set of ordered goals or with a set of *unordered* goals. One might believe there could be two passes through the planner, the first for goal ordering at the planner level, and the second for the detailed assembly at the assembler level with the ordered goals as the input from the first pass. However, to make a right decision about the order of the goals the planner sometimes needs to know the lower level information about which ends and which features of the two parts should go together. It may also need to know the content of the other goals. This requires the planner be able to move among the levels and handle the formation and propagation of those dynamic constraints. Therefore working with ordered goals is not a simple sum of a set of unordered goals and some ordering rules, but instead, an interleaving of goal ordering and feature matching.

4.4 Forward reasoning v. Backward reasoning

The planner developed from a pure forward reasoning model to a pure backward reasoning model, to a combination of both forward and backward reasoning model. It seems that when the planner feels confident about the goal it is going to reach, backward reasoning is faster. When the planner is less confident about what is

going to happen, e.g. when it cannot decide the order for a set of goals in one level, forward reasoning is more natural and efficient.

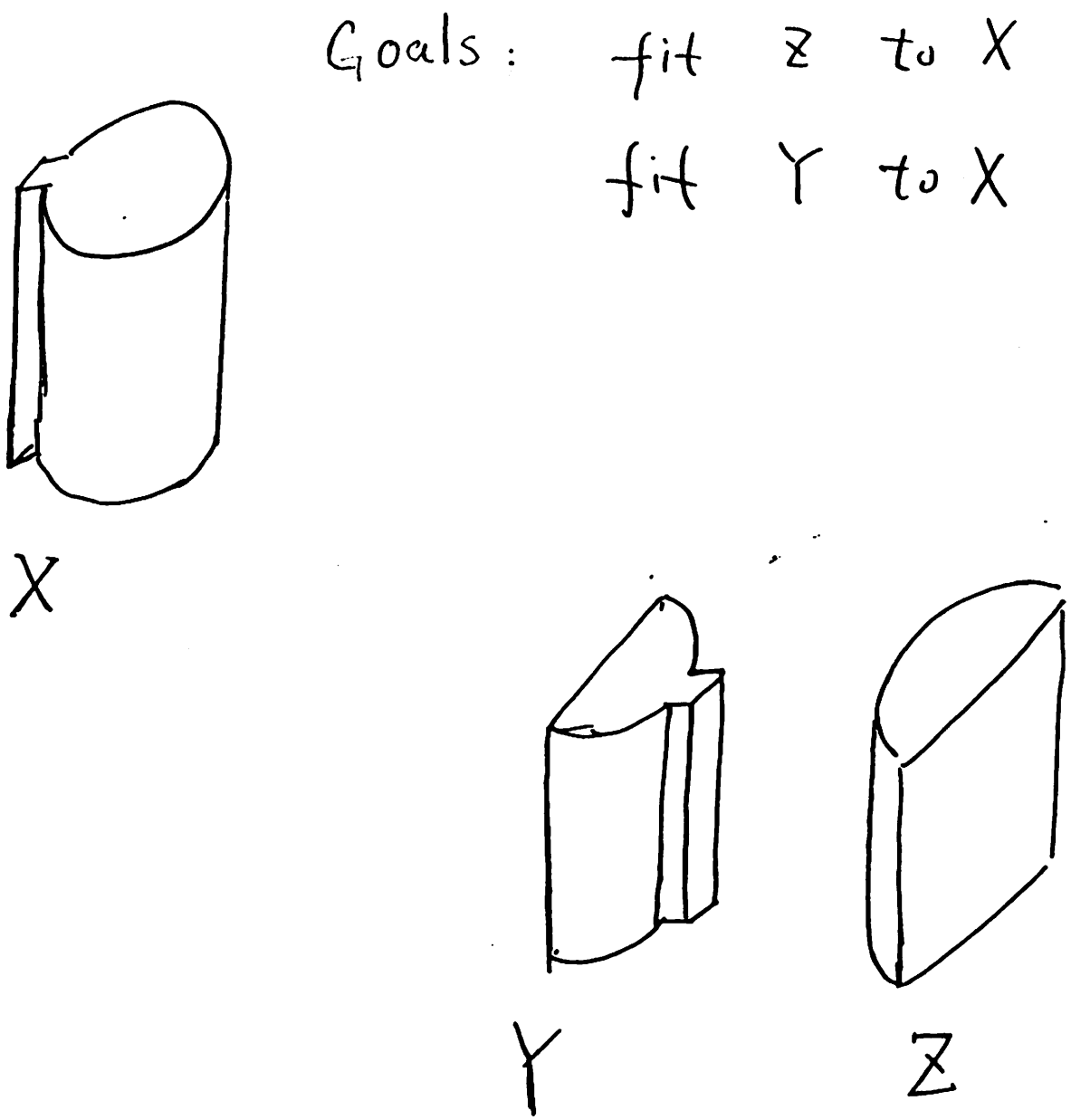


Figure 17: Bob's example

4.5 The combination of control strategy and representation hierarchy

In a simple example, Bob's example (see Figure 17), the planner takes advantage of both the hierarchical representation of the parts and the least commitment planning strategy. The planner stays in a *least commitment* mode as long as it still has some place to find out more information, either deeper (lower level information about the parts) or wider (the content of other goals), or former(the partially assembled result). In this example, when the planner handles the goal "fit part Z to part X", it cannot decide where to put part Z inside part X, since both part X and Z have a feature-β equal to 2π but there is no reason to put part Z in any specific place. The planner delays the decision until after it moves up to the planner level and examines the next goal "fit part Y to part X". After fitting part Y in part X, the decision for part Z's position becomes trivial. A new heuristic rule is created:

IF
The way to assemble the two parts mentioned in the goal cannot be decided uniquely
THEN
Delay the realization of the goal
AND finish those goals which can be decided uniquely first.

This example shows how the control and representation are combined successfully.

5. Conclusions

These are the questions raised in the *Introduction* section:

1. What kind of information about assembly parts is relevant to the assembly domain and how is one to represent that information effectively?;

2. What kind of knowledge of planning and assembly, represented as heuristics and constraints, is needed to make the planner control more efficient?;

3. How is one to combine the information from 1. and 2. together to reach the goal of successfully integrating the strategic, geometric and functional aspects of the assembly problem?

The answers are :

1. The information related to assembly functions is most usefully represented explicitly, and a hierarchical representation for each object seems to be a proper data structure. The planner just extracts up to a certain level of the hierarchy depending on the level of detail needed. This also complements our MOLGEN fashioned hierarchical control structure.

2. In terms of control strategy, an assembly planner can use a least commitment strategy when information is not sufficient, in order to avoid backtracking as much as possible. The search space is limited by relevant information about the assembly parts, such as β and α angles, and the control of assembly feature's birth and death.

3. In order to integrate object representation and control structure, hierarchical planning and hierarchical representation of objects can be an efficient combination which leads the planner to search for relevant information when necessary.

A lot of further work needs to be done. First of all, a more formal schematic decomposition of the planner system should be developed along with the increasing complexity of the system. For example, a learning module may be attached to it for improving the behavior of the planner in the near future. Second, since the output of this planner is supposed to be an input to a lower level robot planner,

the interface in between needs to be worked out concerning the requirements of the lower level planner. Third, the interface between planner and sensors needs to be implemented as the basic step to turn this static planner into a dynamic one. Fourth, lots of ideas discussed previously are waiting for experimentation, such as the proposed representation for feature shape, which needs both more formalization and discussion for some special case processing, different choices for representing related features, and those examples in Figure 15, Figure 16 and Figure 17. And finally, the control structure should be more flexible, more dynamically oriented. Due to characteristics of the planner as said in [2]:

> 1. It incrementally constructs a plan, starting with inadequate information, procuring information and refining the plan as it proceeds.
> 2. Planning and execution may be interleaved; it is not necessary that all of the planning activity precede all of the execution.
> 3. 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.

We feel that the blackboard model of problem solving may be beneficial. In addition to opportunistic reasoning as knowledge-application strategy, the blackboard model prescribes the organization of the domain knowledge and all the input and intermediate and partial solutions needed to solve the problem[11]. The HEARSAY-II blackboard control structure is a potential choice to be combined with the current planner system.

Acknowledgements

We would like to thank Rukmini Vijaykumar, Professor Jack Dixon, Dr. Dan Corkill, Thea Iberall, and Ron Arkin for their helpful suggestions and feedback during the preparation of this paper. Yanxi Liu would also like to thank Robert Collins for his patient work on polishing the English writing of this paper.

References

[1] Cohen, P. R. & Feigenbaum, E. A., Eds. (1982). *The Handbook of Artificial Intelligence*, Vol.3, pp. 513 – 556, William Kaufmann, Inc., Los Altos, California.

[2] Vijaykumar, R., Arbib, M.A. & Liu, Y., (1985). "Dynamic Planning for Sensor-Based Robots". Proceedings of the IFAC *Symposium on Robot Control*, Barcelona, Spain, November 6-8, 1985.

[3] Torras, C. & Thomas, F. (1985). "Planning with Constraints: Application to Sensor-Based Robot Assembly Tasks". Proceedings of the IFAC *Symposium on Robot Control*, Barcelona, Spain, November 6-8, 1985.

[4] Clayton, B (1985). *Automatic Reasoning Tool.* Inference Corporation, 5300 W, Cebtyrt Bkvd,Los Angeles, CA 90045.

[5] G. Boothroyd & P. Dewhurst (1983). *Design for Assembly – A Designer's Handbook* Department of Mechanical Engineering, University of Mass. Amherst, MA.

[6] Stefik, M. (1980). "Planning and Meta-planning(MOLGEN: part2)".*Artificial Intelligence, 6, 141-170.*

[7] CAM-I (1981). "CAM-I's Illustrated Glossary of Workpiece Form Features". Revised May, 1981, R-80-PPP-02.1, CAM-I, Inc., Arlington, Texas.

[8] Stefik, M. (1980)." Planning with Constraints".Ph.D Thesis. Stanford Heuristic Programming Project, memo HPP-80-2, Computer Science Department, Stanford U. Report No. STAN-CS-80-784.

[9] Barr, A. & Feigenbaum, E. (1981). *The Handbook of Artificial Intelligence* Vol.1. pp. 23-24, pp 180-189 William Kaufmann, Inc. Los Altos, California.

[10] Popplestone, R.J. & Ambler, A.P. & Bellos, I. (1978). " A Language for Describing Assemblies". D.A.I. Research Paper No. 79&NEL Colloquium on Robotics, March. Industrial Robot, September 1978.

[11] Nii, H. P. (1986) "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures". AI magazine vol.7, No.2, Summer 1986, *pp. 38-53.*

APPENDIX A

COMPUTER-GENERATED TRACES

The followings are the computer-generated traces. Case one is Huber's example. Case two is Arbib's example. Case three is Yanxi's example and case four is Rukimini's example. The information describes what happens during the procedure the planner solves the problems and tells when and what operator is used and what data updated. The assembler level operator is indicated by "&&&" and the planner level operator by "***".

```
!!!!! Case One !!!!!!!!!
This is the trace for Huber's first example. The input is a set of ordered
goals. The planner figures out how to assemble parts A, B C and D together.
For more description, please refer to the text 3.1 and 3.2.

 => pop
 => reset
Resetting ART...
Knowledge base has been reset.
 => run
Beginning run at 4/09/86 23:15:27: type ^C to halt.


 $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

please enter the number accordingly !
1. Huber's first example
2. Arbib's first example
3. Yanxi's first example
4. Rukmini's first example
5. Dixon's example
6. stop
choice 1


This is the set of goals for Huber's  example ---

fit part B into part A
thread part A to part C
thread part B to part D


goal ordering  is finished !


 **** planner level operator -- goal-focuser
focus attention on the first goal --     fit-together PARTB PARTA


 *** planner level goal refiner refines the goal -- fit PARTB into PARTA -- to :

subgoal one -- match features between PARTB and PARTA

subgoal two -- merge axes of matched features of PARTB and PARTA

subgoal three -- check if the matched features of PARTB and PARTA can be physically engaged

subgoal four -- really fit the matched features of PARTB and PARTA

focus on the first subgoal --  match features


 &&& assembler level operator -- feature matcher

 PARTB END1 FEATURE-1 and PARTA END2 FEATURE-2 are matched

       PARTB       PARTA
matched features FEATURE-1 FEATURE-2
thread NON-THREAD NON-THREAD
shape    rectangular    round
dimension 3.0      >   2.7730849 !


 *** planner level operator : subgoal shifter shifts from
subgoal ( feature-match PARTB PARTA )  to
subgoal ( merge the insertion axes of PARTB and PARTA's matched features )

 &&& assembler level operator -- merge axes
 change the insertion axis of PARTB from X axis  to be the same as PARTA -- along axis Z


 *** planner level operator : subgoal shifter shifts from
 subgoal ( merge-axis PARTB PARTA )  to
 subgoal ( test if the matched features of PARTB and PARTA can be physicaly engaged )

 &&& assembler level operator -- engaged or not checker
there is nothing in the way to assemble PARTB END1 FEATURE-1 and PARTA END2 FEATURE-2 together !


 *** planner level operator : subgoal shifter shifts from
 subgoal ( if-engaged PARTB PARTA )  to
 subgoal ( fit PARTB FEATURE-1 and PARTA FEATURE-2together )
```

&&& assembler level operator -- fit operation

Since no ones' beta equal to zero and both beta values
are equal,
look for the corresponding details of PARTB
END1 FEATURE-1 and PARTA END2 FEATURE-2

Could PARTB END1 FEATURE-1's TWO-CONCAVES fit into PARTA END2 22FEATURE-2's TWO-CONVEXES ?
YES

Since both featuers have beta values equal to 180 degrees
turn around the insertion axes at most 180 degrees to
fit PARTB END1 FEATURE-1 TWO-CONCAVES into PARTA END2 FEATURE-2 NNTWO-CONVEXES !

goal -- fit together PARTB PARTAis done !

*** planner level -- goal shifter(goal-focuser)
shifts from goal < fit-together PARTB PARTA > to goal < thread-in PARTA PARTC >
focus attention on goal < thread-in PARTA PARTC >

*** planner level goal refiner refines the goal -- thread PARTA into PARTC -- to :

subgoal one -- match features between PARTA and PARTC

subgoal two -- merge axes of matched features of PARTA and PARTC

subgoal three -- check if the matched features of PARTA and PARTC can be physically engaged

subgoal four -- really thread the matched features of PARTA and PARTC

focus on the first subgoal -- match features

&&& assembler level operator -- feature matcher

PARTA END2 FEATURE-1 and PARTC END1 FEATURE-1 are matched
matched parts PARTA PARTC
matched features FEATURE-1 FEATURE-1
thread thread thread
shape round round
dimension 3.01 > 3.0

*** planner level operator : subgoal shifter shifts from
subgoal (feature-match PARTA PARTC) to
subgoal (merge the insertion axes of PARTA and PARTC's matched features)

&&& assembler level operator -- merge axes

the insertion axes of the matched features have the same
orientation

*** planner level operator : subgoal shifter shifts from
subgoal (merge-axis PARTA PARTC) to
subgoal (test if the matched features of PARTA and PARTC can be physicaly engaged)

&&& assembler level operator -- engaged or not checker
could PARTA END2 FEATURE-1 and PARTC END1 FEATURE-1 with PARTBww in between be engaged?YES

*** planner level operator : subgoal shifter shifts from
subgoal (if-engaged PARTA PARTC) to
subgoal (thread PARTA FEATURE-1 and PARTC FEATURE-1together)

thread PARTA END2 FEATURE-1 into PARTC END1 FEATURE-1 !

goal -- thread in PARTA PARTCis done !

*** planner level -- goal shifter(goal-focuser)
shifts from goal < thread-in PARTA PARTC > to goal < thread-in PARTB PARTD >
focus attention on goal < thread-in PARTB PARTD >

*** planner level goal refiner refines the goal -- thread PARTB into PARTD -- to :

subgoal one -- match features between PARTB and PARTD

subgoal two -- merge axes of matched features of PARTB and PARTD

subgoal three -- check if the matched features of PARTB and PARTD can be physically engaged

subgoal four -- really thread the matched features of PARTB and PARTD

focus on the first subgoal -- match features

&&& assembler level operator -- feature matcher

PARTB END2 FEATURE-2 and PARTD END1 FEATURE-1 are matched
matched parts PARTB PARTD
matched features FEATURE-2 FEATURE-1
thread thread thread
shape round round
dimension 1.6 > 1.55

*** planner level operator : subgoal shifter shifts from
subgoal (feature-match PARTB PARTD) to
subgoal (merge the insertion axes of PARTB and PARTD's matched features)

&&& assembler level operator -- merge axes
change the insertion axis of PARTB from X axis to be the same as PARTD -- along axis Z

*** planner level operator : subgoal shifter shifts from
subgoal (merge-axis PARTB PARTD) to
subgoal (test if the matched features of PARTB and PARTD can be physicaly engaged)

&&& assembler level operator -- engaged or not checker
there is nothing in the way to assemble PARTB END2 FEATURE-2 and PARTD END1 FEATURE-1 together !

*** planner level operator : subgoal shifter shifts from
subgoal (if-engaged PARTB PARTD) to
subgoal (thread PARTB FEATURE-2 and PARTD FEATURE-1together)

thread PARTB END2 FEATURE-2 into PARTD END1 FEATURE-1 !

!!!!!! Desired state reached !!!!!!
solution found in viewpoint STAGE-38!

```
!!!!!!!!!!!!!!!!!! Case Two !!!!!!!!!!!!!!!!!!!!!
This is the trace for Arbib's first example. The input is a set of ordered
goals. The planner figures out how to assemble parts E, F and G together.
For more description, please refer to the text 3.3.

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

please enter the number accordingly !
1. Huber's first example
2. Arbib's first example
3. Yanxi's first example
4. Rukmini's first example
5. Dixon's example
6. stop
choice 2

This is the set of goals for Arbib's example ---
fit part F into part E
fit part G into part E


goal ordering  is finished !


 **** planner level operator -- goal-focuser
focus attention on the first goal --    fit-together PARTF PARTE


 *** planner level goal refiner refines the goal -- fit PARTF into PARTE -- to :

subgoal one -- match features between PARTF and PARTE

subgoal two -- merge axes of matched features of PARTF and PARTE

subgoal three -- check if the matched features of PARTF and PARTEbb can be physically engaged

subgoal four -- really fit the matched features of PARTF and PARTE

focus on the first subgoal --  match features


 &&& assembler level operator -- feature matcher

 PARTF END1 FEATURE-1 and PARTE END1 FEATURE-2are matched

      PARTF      PARTE
matched features FEATURE-1 FEATURE-2
thread NON-THREAD NON-THREAD
shape    round  round
dimension 1.01 is larger than 1.0


Since PARTF has alpha value equal to 180 degrees
it has two ends identical, just consider the feature(s)
 in one end --> end1 !


 *** planner level operator : subgoal shifter shifts from
subgoal ( feature-match PARTF PARTE )  to
subgoal ( merge the insertion axes of PARTF and PARTE's matched features )

 &&& assembler level operator -- merge axes
change the insertion axis of PARTF from X axis  to be the same as PARTE -- along axis Z


 *** planner level operator : subgoal shifter shifts from
 subgoal ( merge-axis PARTF PARTE )  to
 subgoal ( test if the matched features of PARTF and PARTE can be physicaly engaged )

 &&& assembler level operator -- engaged or not checker
there is nothing in the way to assemble PARTF END1 FEATURE-1 and PARTE END1 FEATURE-2 together !


 *** planner level operator : subgoal shifter shifts from
 subgoal ( if-engaged PARTF PARTE )  to
 subgoal ( fit PARTF FEATURE-1 and PARTE FEATURE-2together )

 &&& assembler level operator -- fit operation
Since one of the features beta value equals to zero
no turn about the insertion axes needed to
fit PARTF END1 FEATURE-1 into PARTE END1 FEATURE-2 !
```

goal -- fit together PARTF PARTEis done !

*** planner level -- goal shifter(goal-focuser)
shifts from goal < fit-together PARTF PARTE > to goal < fit-together PARTG PARTE >
focus attention on goal < fit-together PARTG PARTE >

*** planner level goal refiner refines the goal -- fit PARTG into PARTE -- to :

subgoal one -- match features between PARTG and PARTE

subgoal two -- merge axes of matched features of PARTG and PARTE

subgoal three -- check if the matched features of PARTG and PARTEbb can be physically engaged

subgoal four -- really fit the matched features of PARTG and PARTE

focus on the first subgoal -- match features

&&& assembler level operator -- feature matcher

PARTG END1 FEATURE-1 and PARTE END1 FEATURE-3are matched

	PARTG	PARTE
matched features	FEATURE-1	FEATURE-3
thread	NON-THREAD	NON-THREAD
shape	round	round

dimension 1.01 is larger than 1.0

Since PARTG has alpha value equal to 180 degrees
it has two ends identical, just consider the feature(s)
in one end --> end1 !

*** planner level operator : subgoal shifter shifts from
subgoal (feature-match PARTG PARTE) to
subgoal (merge the insertion axes of PARTG and PARTE's matched features)

&&& assembler level operator -- merge axes
change the insertion axis of PARTG from X axis to be the same as PARTE -- along axis Z

*** planner level operator : subgoal shifter shifts from
subgoal (merge-axis PARTG PARTE) to
subgoal (test if the matched features of PARTG and PARTE can be physicaly engaged)

&&& assembler level operator -- engaged or not checker
there is nothing in the way to assemble PARTG END1 FEATURE-1 and PARTE END1 FEATURE-3 together !

*** planner level operator : subgoal shifter shifts from
subgoal (if-engaged PARTG PARTE) to
subgoal (fit PARTG FEATURE-1 and PARTE FEATURE-3together)

&&& assembler level operator -- fit operation
Since one of the features beta value equals to zero
no turn about the insertion axes needed to
fit PARTG END1 FEATURE-1 into PARTE END1 FEATURE-3 !

!!!!!! Desired state reached !!!!!!
solution found in viewpoint STAGE-661

```
!!!!!!!!!!!!!!!! Case Three !!!!!!!!!!!!!!!!!!!!!!!!!!!!
This is the trace for Yanxi's example. The input is a set of ordered
goals. The planner figures out how to assemble parts H, I and J together.
For more description, please refer to the text 3.4.

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

please enter the number accordingly !
1. Huber's first example
2. Arbib's first example
3. Yanxi's first example
4. Rukmini's first example
5. Dixon's example
6. stop
choice 3

This is the set of goals for Yanxi's example ---
fit part I into part H
fit part J into part H


goal ordering  is finished !


 **** planner level operator -- goal-focuser
focus attention on the first goal --    fit-together PARTI PARTH


 *** planner level goal refiner refines the goal -- fit PARTI into PARTH -- to :

subgoal one -- match features between PARTI and PARTH

subgoal two -- merge axes of matched features of PARTI and PARTH

subgoal three -- check if the matched features of PARTI and PARTHbb can be physically engaged

subgoal four -- really fit the matched features of PARTI and PARTH

focus on the first subgoal --  match features


 &&& assembler level operator -- feature matcher

 PARTI END1 FEATURE-1 and PARTH END1 FEATURE-1are matched

      PARTI      PARTH
matched features FEATURE-1 FEATURE-1
thread NON-THREAD NON-THREAD
shape   half-round  round
dimension 1.01 is larger than 1.0

 !!!! new feature is born at PARTHEND1
as container with
          beta equal to 360 and shape half-round !!!!


Since PARTI has alpha value equal to 180 degrees
it has two ends identical, just consider the feature(s)
 in one end --> end1 !


 *** planner level operator : subgoal shifter shifts from
subgoal ( feature-match PARTI PARTH )  to
subgoal ( merge the insertion axes of PARTI and PARTH's matched features )

 &&& assembler level operator -- merge axes
 change the insertion axis of PARTI from X axis  to be the same as PARTH -- along axis Z


 *** planner level operator : subgoal shifter shifts from
 subgoal ( merge-axis PARTI PARTH )  to
 subgoal ( test if the matched features of PARTI and PARTH can be physicaly engaged )

 &&& assembler level operator -- engaged or not checker
there is nothing in the way to assemble PARTI END1 FEATURE-1 and PARTH END1 FEATURE-1 together !


 *** planner level operator : subgoal shifter shifts from
 subgoal ( if-engaged PARTI PARTH )  to
 subgoal ( fit PARTI FEATURE-1 and PARTH FEATURE-1together )

 &&& assembler level operator -- fit operation
```

```
Since one of the features beta value equals to zero
no turn about the insertion axes needed to
fit PART1 END1 FEATURE-1 into PART# END1 FEATURE-1 !


goal -- fit together PART1 PART#is done !



*** planner level -- goal shifter(goal-focuser)
                    shifts from goal < fit-together PART1 PART#   >  to goal < fit-together PARTJ PART# >
focus attention on goal < fit-together PARTJ PART# >

 *** planner level goal refiner refines the goal -- fit PARTJ into PART# -- to :

subgoal one -- match features between PARTJ and PART#

subgoal two -- merge axes of matched features of PARTJ and PART#

subgoal three -- check if the matched features of PARTJ and PART#bb can be physically engaged

subgoal four -- really fit the matched features of PARTJ and PART#

focus on the first subgoal --  match features


 &&& assembler level operator -- feature matcher

 PARTJ END1 FEATURE-1 and PART# END1 NEW-FEATURE-1are matched

      PARTJ  PART#
matched features FEATURE-1 NEW-FEATURE-1
thread NON-THREAD NON-THREAD
shape    half-round  half-round
dimension 1.01 is larger than 1.0


Since PARTJ has alpha value equal to 180 degrees
it has two ends identical, just consider the feature(s)
 in one end --> end1 !


 *** planner level operator : subgoal shifter shifts from
subgoal ( feature-match PARTJ PART# )  to
subgoal ( merge the insertion axes of PARTJ and PART#'s matched features )

 &&& assembler level operator -- merge axes
 change the insertion axis of PARTJ from X axis  to be the same as PART# -- along axis Z


 *** planner level operator : subgoal shifter shifts from
 subgoal ( merge-axis PARTJ PART# )  to
 subgoal ( test if the matched features of PARTJ and PART# can be physicaly engaged )

 &&& assembler level operator -- engaged or not checker
there is nothing in the way to assemble PARTJ END1 FEATURE-1 and PART# END1 NEW-FEATURE-1 together !


 *** planner level operator : subgoal shifter shifts from
 subgoal ( if-engaged PARTJ PART# )  to
 subgoal ( fit PARTJ FEATURE-1 and PART# NEW-FEATURE-1together )

 &&& assembler level operator -- fit operation


Since no ones' beta equal to zero and both beta values
                    are equal,
look for the corresponding details of PARTJ
END1 FEATURE-1 and PART# END1 NEW-FEATURE-1


Since both features have beta values equal to 360 degrees
turn around the insertion axes at most 360 degrees to
fit PARTJ END1 FEATURE-1 HALF-ROUND into PART# END1 NEW-FEATURE-1 --HALF-ROUND !


 !!!!!!   Desired state reached !!!!!!
 solution found in viewpoint STAGE-97!
```

```
!!!!!!!!!!!!!!!!!!!! Case Four !!!!!!!!!!!!!!!!!!!!!!!!!!
This is the trace for Rukmini's example. The input is a set of unordered
goals. The planner figures out how to assemble parts K, M and N together
in a right order. For more description, please refer to the text 3.5.

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

please enter the number accordingly !
1. Huber's first example
2. Arbib's first example
3. Yanxi's first example
4. Rukmini's first example
5. Dixon's example
6. stop
choice 4

This is the set of goals for Rukmini's example ---
fit part M into part K
fit part N into part M


 goal fit-together PARTN PARTM before goal fit-together PARTM PARTK


goal ordering  is finished !


 **** planner level operator -- goal-focuser
focus attention on the first goal --    fit-together PARTN PARTM


 *** planner level goal refiner refines the goal -- fit PARTN into PARTM -- to :

subgoal one -- match features between PARTN and PARTM

subgoal two -- merge axes of matched features of PARTN and PARTM

subgoal three -- check if the matched features of PARTN and PARTMbb can be physically engaged

subgoal four -- really fit the matched features of PARTN and PARTM

focus on the first subgoal --  match features


 &&& assembler level operator -- feature matcher

 PARTN END1 FEATURE-2 and PARTM END1 FEATURE-1are matched

      PARTN      PARTM
matched features FEATURE-2 FEATURE-1
thread NON-THREAD NON-THREAD
shape    round  round
dimension 1.0 is larger than 0.82

 *** planner level operator : subgoal shifter shifts from
subgoal ( feature-match PARTN PARTM )  to
subgoal ( merge the insertion axes of PARTN and PARTM's matched features )

 &&& assembler level operator -- merge axes

the insertion axes of the matched features have the same
                    orientation


 *** planner level operator : subgoal shifter shifts from
 subgoal ( merge-axis PARTN PARTM )  to
 subgoal ( test if the matched features of PARTN and PARTM can be physicaly engaged )

 &&& assembler level operator -- engaged or not checker
there is nothing in the way to assemble PARTN END1 FEATURE-2 and PARTM END1 FEATURE-1 together !


 *** planner level operator : subgoal shifter shifts from
 subgoal ( if-engaged PARTN PARTM )  to
 subgoal ( fit PARTN FEATURE-2 and PARTM FEATURE-1together )

 &&& assembler level operator -- fit operation
Since one of the features beta value equals to zero
no turn about the insertion axes needed to
fit PARTN END1 FEATURE-2 into PARTM END1 FEATURE-1 !
```

```
goal -- fit together PARTN PARTMis done !


*** planner level -- goal shifter(goal-focuser)
                    shifts from goal < fit-together PARTN PARTM   >  to goal < fit-together PARTM PARTK >
focus attention on goal < fit-together PARTM PARTK >

 *** planner level goal refiner refines the goal -- fit PARTM into PARTK -- to :

subgoal one -- match features between PARTM and PARTK

subgoal two -- merge axes of matched features of PARTM and PARTK

subgoal three -- check if the matched features of PARTM and PARTKbb can be physically engaged

subgoal four -- really fit the matched features of PARTM and PARTK

focus on the first subgoal --  match features


 &&& assembler level operator -- feature matcher

 PARTM END1 FEATURE-2 and PARTK END1 FEATURE-1are matched

       PARTM      PARTK
matched features FEATURE-2 FEATURE-1
thread NON-THREAD NON-THREAD
shape    round  round
dimension 2.0 is larger than 1.02

 *** planner level operator : subgoal shifter shifts from
subgoal ( feature-match PARTM PARTK )  to
subgoal ( merge the insertion axes of PARTM and PARTK's matched features )

 &&& assembler level operator -- merge axes

the insertion axes of the matched features have the same
                    orientation


 *** planner level operator : subgoal shifter shifts from
 subgoal ( merge-axis PARTM PARTK )  to
 subgoal ( test if the matched features of PARTM and PARTK can be physicaly engaged )

 &&& assembler level operator -- engaged or not checker
there is nothing in the way to assemble PARTM END1 FEATURE-2 and PARTK END1 FEATURE-1 together !


 *** planner level operator : subgoal shifter shifts from
 subgoal ( if-engaged PARTM PARTK )  to
 subgoal ( fit PARTM FEATURE-2 and PARTK FEATURE-1together )

 &&& assembler level operator -- fit operation
Since one of the features beta value equals to zero
no turn about the insertion axes needed to
fit PARTM END1 FEATURE-2 into PARTK END1 FEATURE-1 !


 !!!!!!  Desired state reached !!!!!!
 solution found in viewpoint STAGE-122!


 $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

please enter the number accordingly !
1. Huber's first example
2. Arbib's first example
3. Yanxi's first example
4. Rukmini's first example
5. Dixon's example
6. stop
choice 6
OK, that's it ! Bye !

goal -- fit together PARTM PARTKis done !

 [Abort]
 => [Abort]
```