Empirical Assembly Planning: A Learning Approach

Heedong Ko

MLI 88-18

CHAPTER 1.

INTRODUCTION

This thesis explores various problems that arise when planning for constructing an assembly structure from its structural description. The purpose of this exploration is twofold. One is to bring the manufacturing considerations to bear during the design phase. Currently, a designer of an assembly structure, an assembly designer, receives little feedback from the manufacturing engineer during the design phase whether the designed assembly structure is assemblable. An automatic assembly planning system can provide such a feedback from which the assembly designer may improve the structural model immediately rather than having to wait until the manufacturing phase of the designed product.

The other purpose is to study the issues in building a more autonomous system, interacting with the physical environment, as an avenue of conducting Artificial Intelligence (AI) research. The physical environment can be viewed as a spatially and temporally structured object. Its scenery is comprised of the physical objects at different positions and having its own internal structure. Furthermore, the scenery changes in time due to some physical processes of the environment, maybe initiated by an action. A sequence of changes of the scenery constitutes a temporal object. In order to perceive and interact with the environment, an intelligent agent needs to be able to build and use descriptions of the spatial and temporal objects. Furthermore, these two types of descriptions are inexorably intertwined with each other.

To pursue the goal of constructing an assembly structure, the planner needs a description of the assembly structure as a goal statement, a spatially structured description. Then, the planner produces the plan of pursuit as a sequence of assembly steps. A sequence of assembly steps describes an assembly process, a temporally structured object. Then, the act of assembly planning is building a

relationship between these two types of descriptions. Therefore, planning for an assembly structure and learning from the assembly planning experiences require solving the pertinent issues in building an autonomous intelligent machine that perceives and interacts with the physical environment.

To limit the scope of the problem, this thesis research does not explore the object manipulation in favor of the more impending research issues for the assembly planning. This subject is a focus of modern robotics research and we do not have to start from scratch when carrying out interfacing with a manipulator when the assembly planning problem is expanded to consider object manipulation. In fact, the geometric representation of the environment, adopted in this research, is based on the world modeling component of many proposed intelligent robotic systems [Fahlman 1974; Wesley & Lozano-Perez & Lieberman, 1980]. An assembly task domain is so complex that it requires conglomerate research efforts from different areas; therefore, one of the main concerns of this research is their interfaces and hence, the geometric modeling system is used to model the environment.

1.1. Problem Specification

An assembly is a collection of components that are brought together to form a connected structure. A component occupies a fixed volume of space, and its shape does not change, a rigid body: i.e., no deformation or breakage of the components are allowed. A pair of components may be brought together to establish a kinematic connection. Objects connected by non-kinematic structural connections (those achieved by gluing, welding, or squeezing, for example) are not considered as separate components of an assembly.

The term, "assembly", may denote the structure resulting after an assemblage or the action of the assemblage although the word "assembly" does not have a verb form in the dictionary. Here, we use "target assembly" for the structural specification of the assembly and "mating operation" for the

assembly action bringing together two structures to form their kinematic connections simultaneously. The objective of the assembly planner is to generate an assembly sequence, consisting of mating operations, so that there is a collision-free path for each mating operation.

1.1.1. Blocks World

Traditionally, the difficulty of the mechanical assembly domain has been controlled by restricting the number of different predicates and axioms. For example, in a blocks world, only a few primitive symbolic predicates are known: "ON" for a spatial relationship, "ON-TOP-OF" for spatial occupancy, and "SUPPORT" for stability. This approach has two drawbacks. First, too many aspects of the physical world are being modeled without an adequate fidelity [Hayes, 1979]. The physical world is governed by dynamics: spatial, kinematic, and kenetic attributes (this thesis models only spatial and kinematic attributes). In order to capture all these aspects of the physical world, the model is diluted at the expense of a broad coverage of all these laws. As a consequence, although a blocks world domain concerns assemblies, this domain model is virtually useless when extending the model to cover more realistic mechanical assemblies, consisting of components with complex shapes and many linkages.

In this thesis, the difficulty is controlled by a simulation. The physical environment for the assembly actions is simulated with a small subset of the physical laws. This approach is to sacrifice the broad coverage at the expense of the increased coverage of those laws of the simulated environment.

The second drawback is that the domain is modeled at such a high level of abstraction that it is hard to devise an interface to the real world without arbitrary assumptions that are completely hidden from the reasoner of the domain model. As a result, the interface between the high level reasoner and the robot is achieved with arbitrary pieces of knowledge for the task at hand that are too little

understood to lay a foundation for future work. The environment should be modeled in a graphic detail so that an interface with the real world is not such a paramount difficulty in itself. Only a few AI researcher felt that this was necessary [Fahlman, 1974; Popplestone & Ambler & Bellos, 1980; Segre, 1987] for robotic assembly.

1.1.2. Scope of the Implementation

The thesis research was studied by implementing a computer program. The program is called NOMAD for NOrmative Mechanical Assembly Device. NOMAD covers a broad area from assembly design to planning *vertically*. NOMAD integrates from the assembly design phase to planning and learning aspects of the assembly planning, a vertical approach.

A horizontal approach would be to solve a single problem completely among the many problems in the area of the assembly design and planning. The horizontal approach to a problem domain would be appropriate if the structure of the problem domain is already well-understood: the isolatable subproblems and their interrelationships. However, this is hardly the case with the assembly task domain. A vertical approach has been taken to uncover the structure of the assembly task domain and to study the synergistic interrelationships between the uncovered subproblems.

A ramification of this vertical approach is that individual subproblems uncovered during the course of the investigation may not be completely solved. In particular, many mechanisms developed for NOMAD are incomplete and limited to a small variety of types of cases because the completeness of the mechanisms was sacrificed when a new problem important toward a vertical integration was uncovered and the current problem was well-enough understood so that addressing the new problem immediately would not have caused a sever breakage in the vertical integration. On the other hand, the new problem is bypassed when

(1) the new problem was being addressed by other researchers and it was unlikely that investigating them would unveil any new insights without launching a project for another PhD dissertation: e.g., trajectory planning (to be discussed), and

(2) the new problem could be solved easier when the input information is augmented: e.g., a sensory feedback for reasoning about ranges.

1.2. Overview of the Approach

An assembly designer models the shapes of the components (or simply components) using a geometric modeling system (GMS) and places them in the environment by specifying their initial positions. The components and their positions comprise the scene of an environment that is represented by a *position map*. Then, he designs the target assembly by specifying the components and their kinematic connections.

The kinematic connections are specified symbolically in terms of the *mating conditions* achieved by a mating operation: the local spatial and kinematic constraints between the geometric features of the components. The assembly planner must envision the possible configurations after mating operations in order to project the effects of planning a sequence of mating operations. This thesis develops the Spatial Reasoning Engine (SRE) that infers the relative positions of the objects in multiple reference frames, their kinematic degrees of freedom, methods for combining local spatial constraints between features of the objects, and distances.

To assemble the target assembly, all the specified kinematic connections should be achieved so that the mating operations do not interfere with each other. This is ensured by imposing precedence constraints between the mating operations. The precedences are imposed by detecting spatial interferences of the swept volume by a mating operation with the occupied space by the components already assembled, forming a subassembly. This thesis presents a "first-principle" planning method

based on the projection of the ill-effects of the spatial interferences.

Unfortunately, the "first-principle" plan generator may have to examine all the possible sequences of mating operations to discover all the precedence constraints, a combinatorially explosive process. To limit the number of spatial interference checkings, the target assembly is hierarchically organized, forming an *assembly hierarchy*. Using the assembly hierarchy as an *abstraction hierarchy* of the assembly planning problem, the assembly planner may reduce the number of spatial interference checkings by limiting the spatial interferences within each subassembly of the hierarchy. Note, this strategy assumes that the assembly hierarchy is a "correct" abstraction hierarchy. Therefore, the saving by constructing the assembly hierarchy is conditional on the correctness of the hierarchy as an abstraction hierarchy.

In order to guarantee the correctness of an abstraction hierarchy, an exhaustive spatial interferences between sibling subassemblies should checked, losing the savings. Therefore, there are no savings by constructing the assembly hierarchy unless there exist a theory of constructing a "correct" abstraction hierarchy from the target assembly specification or the system can somehow bring previously checked assembly experience to bear in constructing an assembly hierarchy of a new target assembly. Here, a heuristic method for constructing an assembly hierarchy is presented for the former case; a learning method is presented for the latter case.

Instead of building a fool-proof assembly hierarchy initially, the learning method constructs an assembly hierarchy from the memory of previously constructed assembly plans and their target assemblies. Assuming the assembly hierarchy is correct, the assembly planner generates the final assembly sequence from the hierarchy. However, when the assembly hierarchy is incorrect, it is detected and the planner assimilates the failure by limiting the scope of the application of the assembly plans and their target assemblies. When successful, the planner may extend the scope of the applied assembly plans and their target assemblies. The thesis presents a framework for learning by

assimilating new assembly planning experiences and planning hierarchically by applying them in new assembly task situations.

The assembly planning experiences are assimilated as the groupings and the precedence relations between their components. A grouping is a collection of components like an assembly structure but it can be assembled by fixing one component, called a base, and mating the rest of the components in some sequence, an assembly subsequence. The assembly hierarchy construction is divided into two steps: forming groupings of components and organizing them into subassemblies. Then, the assembly subsequences from the groupings in the hierarchy are resolved into an overall assembly sequence.

This resolution requires inference of precedence relations. The sequence of mating operations from some of the groupings may be inconsistent. This thesis develops a *precedence calculus* to detect inconsistencies and to derive additional precedences.

1.3. Contributions

This research confronts the problems in integrating multiple heterogeneous areas that would not have come up if each area were studied in isolation. This section will examine these problems and their solutions abstractly from the view of the overall organization of the thesis research before delving into their technical discussions in the rest of the thesis.

1.3.1. Position Map

It is necessary that the assembly planner must envision many possible subassembly states as scenes of the environment. If each new scene is represented distinctly after a mating operation, it would be wasteful because most of the positions of objects are not changed after the operation.

Furthermore, given a scene, the number of relative positions is exponential. Consider each geometric feature as a node and a link between nodes as their relative positions. Then, the number of possible relative positions would be $\binom{N}{2}$ where N is the number of geometric features. Only a small number of them (typically one to three) are relevant to the mating operation. Therefore, multiple scenes should be represented efficiently and the relative positions should remain implicit in the scene representation until they are required.

This thesis develops a position map. A node in the position map is a geometric feature and each directed link from node A to B is labeled with a homogeneous transformation matrix (explained in the next chapter) representing relative spatial positions of A with respect to B. A graph spanning algorithm is used to compute relative positions between two nodes of the graph. Because all the geometric features of an object are positioned with respect to its *base frame*, only the links from the inertial frame of the world to the base frames of the objects in the environment are needed to represent the scene. When the number of components are within a reasonable number, all the base links can be used to represent each scene.

1.3.2. Spatial Reasoning Engine

Kinematic connections between components of an assembly structure are specified by the assembly designer in terms of their mating conditions. Therefore, a mating condition should be easy to use and based on the notions familiar to the designer. Possible mating conditions include an *against* condition between two planar faces and an *aligned* condition between two axes.

1.3.2.1. Layered Computation

A mating condition between two object features constrain the relative positions between two objects. SRE determines if the constraints from each mating condition can be composed by a special

purpose matching procedure, a linear numerical method, or a full-fledged non-linear numerical technique. Furthermore, it determines the extent of the freedom of motion between them to maintain contacts between the two geometric features being mated.

1.3.2.2. Conditional Effects in Applying a Mating Operator

Previously [Fikes & Nilsson, 1971], an operator application transforms the current world model by deleting and adding literals in the world model. The added literals constitute the add-list of the operator and deleted literals constitute delete-list of the operator. The literals are added and deleted independently. However, this assumption may fail due to *conditional effects*: the effects of applying an operator are dependent on the other literals in its add-list and the literals in its delete-list.

Here, a mating operator may achieve multiple mating conditions at the same time. The overall effect of the mating operation is a function of the local effects of achieving individual mating conditions and the current positions of the components. The displacement and rotations to achieve a mating condition are dependent on the current locations and orientations of the mating geometric features. SRE combines the displacements and rotations from each mating condition. This processing is dependent on all the mating conditions and the current geometric mating features simultaneously.

1.3.3. Assembly Hierarchy

The assembly planner organizes the goal statement involving the mating conditions of the target assembly into an assembly hierarchy. This goal hierarchy represents structuring the problem into levels of the abstractions so that given a level, only the goals at that level are considered for the planning, ignoring the goals at lower levels of abstraction.

A traditional planner constructs a hierarchy by the subgoaling method of the means-ends analysis: that is, given a conjunction of individual goals, a goal statement, an operator is planned that would satisfy some of the individual goals. If some preconditions of the operator are not met, they become part of the goal statement; this process recurs to form a subgoal hierarchy.

1.3.3.1. Two-Fold Assembly Hierarchy Construction via Grouping

Components of a target assembly are collected into separate groupings prior to constructing an assembly hierarchy. A grouping participates in constructing an assembly hierarchy as a subassembly, but the difference between a grouping and a subassembly is that a subassembly is formed in the context of an assembly hierarchy while a grouping is context independent. As a result, the assembly hierarchy construction is divided into grouping formation and hierarchy formation. The latter requires an "Abstraction Calculus". This thesis does not present any general-purpose solution to this problem. However, the distinction between the grouping and the hierarchy formations is made because the former requires recalling mechanism while the latter requires a inference mechanism. Currently, the assembly hierarchy is formed semi-automatically: the system may query the user to form the assembly hierarchy for complicated assemblies.

The benefit of forming an intermediate structure like a grouping before constructing an assembly hierarchy from the target assembly is threefold. First, the assembly experience is broken into smaller pieces for each grouping rather than it is remembered as a single episode with one gigantic assembly hierarchy. This increases the applicability of the assembly experiences in a new assembly task situation because only their groupings needs to bear resemblance to the portion of the target assembly to be remembered. Second, a gigantic hierarchy that is rarely applicable is a waste of storage to memorize. A grouping takes up less space and, therefore, is less expensive to keep. The final point is a conjecture: it seems we are not good at memorizing a complicated structure like an

assembly hierarchy at one gulp. This hierarchy is the manifestation of accumulating the previous *chunks* into a new *chunk*: note this definition is recursive.

1.3.3.2. Liaison-based Assembly Hierarchy Representation

Previously, an assembly hierarchy was represented by a component hierarchy [Ko & Lee, 1987]: a subassembly is a collection of subassemblies and components. When a component belongs to more than one subassembly, the component hierarchy yields an entangled hierarchy. This is a problem for the planner that uses an assembly hierarchy as a goal hierarchy because multiple tasks in the subassemblies are interacting through the shared components. Here, the hierarchy is constructed based on the pair-wise relations, the kinematic liaisons (to be explained later). When the subassemblies do not share the kinematic liaisons, the liaison-based representation forms a tree-like structure for the assembly hierarchy even if they share components.

1.3.4. Precedence Calculus

In general, it is necessary to specify a precedence between the boolean combinations of the mating operations. For example, "Mate part A with B before mating part C with B or D with B". Moreover, all the precedence relations may be inconsistent when they are posted from the empirically learned experience of the precedence constraints because the learning process may yield faulty concepts for a precedence relation. Therefore, the constraint propagation technique [Allen, 1983] for temporal reasoning is not sufficient. Instead, the precedence calculus is developed to reason from the precedence relations between boolean combination of mating operations and can detect inconsistencies so that it can detect the maximal environment under which all the precedences are consistent.

1.3.5. Conjunctive Goal Interaction via Collision Physics

Domain-independent planners are concerned with the detection and repair of the conjunctive goal interaction [Waldinger, 1977]. There, the focus was to detect interactions in the plan structure by analyzing what became true and false after a hypothetical operator application (add and delete lists) and reorder them to avoid such interactions. Therefore, their method was domain-independent but required add and delete lists of the operators of the domain.

Unfortunately, it is impossible to detect the interferences from the add and delete lists between mating operators because the interaction is caused by the swept volume *during* the mating operation that is not amenable to add and delete list style of operator modeling. Therefore, the previous conjunctive goal interaction method is useless. Rather, this thesis presents the conjunctive goal interaction detection by projecting the effects of constructing a partially assembled structure with the remaining mating operations. Hence, this approach to conjunctive goal interaction does not rely on the add and delete lists of an operator but on the physics of the environment, collision physics.

1.3.6. Empirical Assembly Planning and Learning

The collision physics is a complete and sound theory of the assembly task domain. However, it is intractable in general to apply the theory directly because collision avoidance problem is exponential [Donald, 1984]. Alternatively, this thesis develops a learning method that builds a memory structure that is *empirically sound* and *complete* for the scope of the tasks experiences so far as an *approximate task theory*. This empirical knowledge is built by accumulating the macro-operators empirically; unlike STRIPS [Fikes & Hart & Nilsson, 1972], the macro-operators are accumulated analytically using Explanation-based Learning method [Segre, 1987].

1.3.6.1. Combining Spatial and Temporal Structure Learning

The physical environment can be viewed as a spatially and temporally structured object. Its scenery is comprised of the physical objects at different positions. They are spatially structured. Furthermore, the scenery changes with time due to physical processes occurring in the environment. A sequence of changes of the scenery constitutes a temporal object. In order to perceive and interact with the environment, the domain theory as a model of the environment should include both spatial and temporal concepts.

Learning structural concepts has been a long standing research problem in machine learning. So far, learning from spatially structured objects and learning from temporally structured objects have been studied independently. Initially, the former concentrated on domain-independent methods for learning structural concepts [Winston, 1970; Larson, 1977; Stepp, 1984]. Recently, the latter concentrated on domain-independent methods for learning from a temporal sequence of objects, qualitative process prediction [Michalski & Ko & Chen, 1987; Chen, 1988]. This research focuses on learning the structural descriptions involving both the spatial and temporal relationships because of the following conjecture: the constraints from the spatial and temporal information are inexorably inter-related and should be studied together.

To generalize from multiple spatial structures, the objects from each structure should be mapped with the corresponding objects from other structures. In general, the problem requires solving subgraph isomorphism problem: NP-complete. However, the spatial structures in this thesis, called groupings, are associated with the sequence of mating operators as a planning episode. Mapping between the groupings can proceed after mapping between their associated sequences of mating operators. This problem of mapping two sequences is polynomially bound. This thesis demonstrates that the subgraph isomorphism problem can be overcome when taking advantage of the interaction between spatial and temporal constraints of the physical environment.

1.3.6.2. Constructive Induction

A constructive induction is a form of induction that generates hypotheses including descriptors not present in the input data but constructed by the system [Michalski 1983]. NOMAD develops a novel constructive induction method. The method is based on *deductive restructuring* of the training instances to generate new "interesting" instances that are not covered by the current working hypotheses in memory so that the system may shift its focus of attention from the working hypothesis to a new hypothesis guided by the new instances.

1.3.6.3. Constructive Closed-loop Learning

The learning strategy used by NOMAD is that the planner may start from incorrect and incomplete task theory but can modify and augment its current task theory and hopefully, the process converges to a tractable, correct and complete task theory with sufficient experience. This requires more powerful learning strategy than any one learning mechanism like empirical inductive learning (EIL) and explanation-based learning (EBL).

A *constructive closed-loop learning* (CCL) is a unifying learning strategy that subsumes both EIL and EBL [Michalski & Ko, 1988; Kodratoff & Michalski, 1989]. A CCL mechanism adapts its inference modes to the learning situation according to the relevance of its prior knowledge: the CCL mechanism may find background knowledge using a deductive inference when it logically implies the observation of interest; otherwise, it resorts to an inductive inference to find an additional (or modified) knowledge that would explain the observation. NOMAD is one of the first implementations of the CCL learning paradigm.

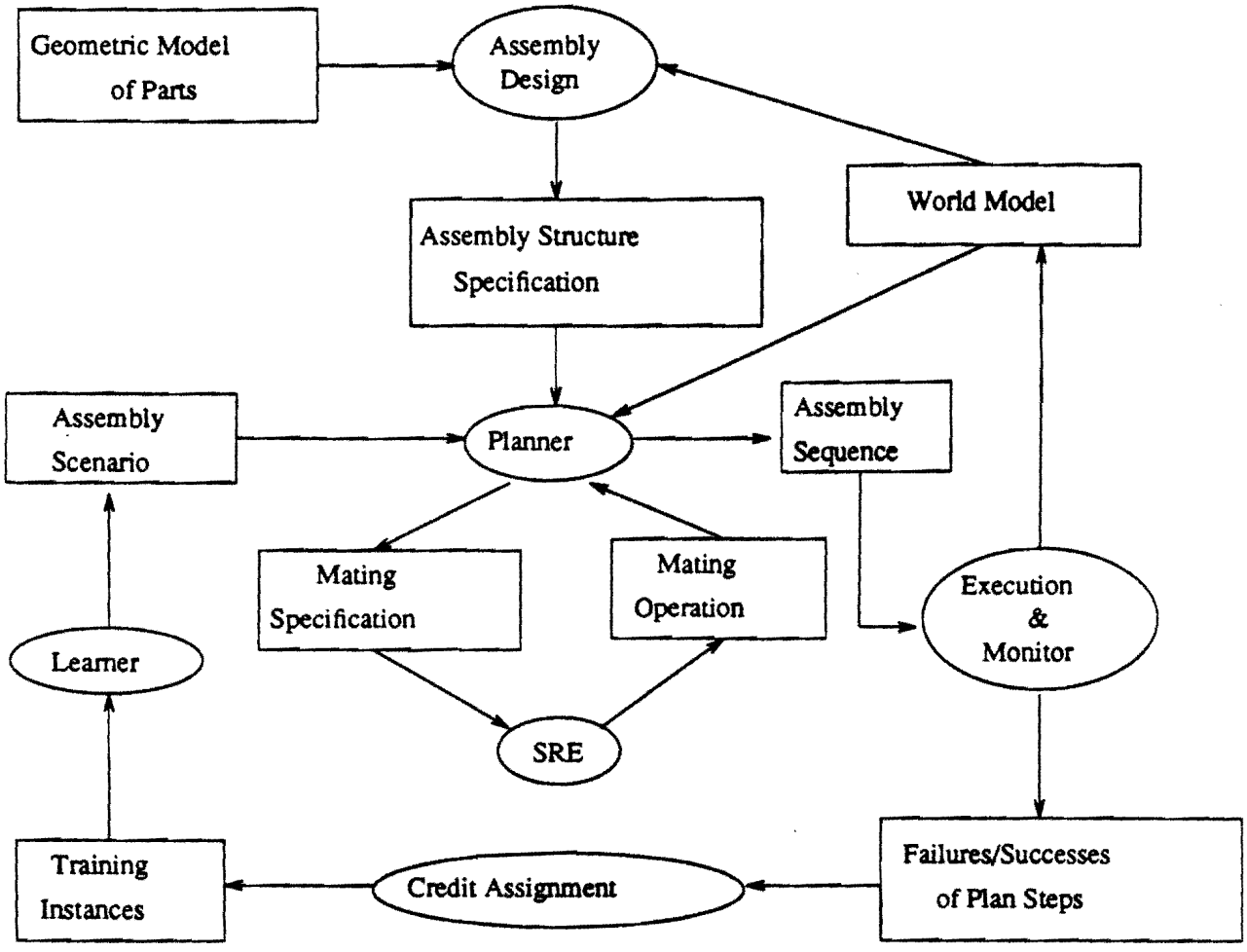


Figure 1.1 NOMAD Architecture

1.4. Outline of the Thesis

The thesis is organized following that of NOMAD. Fig. 1.1 shows the overall architecture of the system. Each oval is a processing element and the rectangles are data elements. Chapter 2 describes the world modeling through a graphic simulation of the environment. Chapter 3 describes

the assembly design in terms of the mating conditions, providing the specification of the target assembly. Chapter 4 describes the spatial reasoning engine (SRE) that can compute the relative positions of the components satisfying the mating conditions simultaneously.

An assembly planning is carried out in two ways: one is projection-based and the other is experience-based. Chapter 5 describes a projection-based approach to an assembly planning: it detects goal interactions by a spatial envisionment of mating operations and posts constraints to avoid such interaction as a precedence between mating operations. Chapter 6 describes a planning method from planning experiences stored in memory as well as the plan execution and monitoring for the credit assignment for the learner. Chapter 7 describes the learning procedure that accumulates the planning experiences from the planning episodes.

CHAPTER 2.

ENVIRONMENT MODELING

An assembly operation effects the change in the position of an object in the environment and the change is determined by the set of mating conditions to be satisfied and the current position of the object. Hence, the correct perception of the position of objects in the environment is crucial to the success of an assembly operation. The perception internalizes the environment into an internal description in its memory, a *world model* [Fikes & Nilsson 1971]; this internalization process from sensory input is beyond the scope of this thesis research.

This process is simulated by user-defined interpretive routines: they internalize the graphical description of the environment by translating the description into a memory structure. Fig. 2.1 shows two distinctions. A scene modeler models the shapes of objects and their positions in the physical environment (its snapshot) by a geometric modeling system (GMS). A world modeler internalizes the graphic representation of the scene into the memory as a position map.

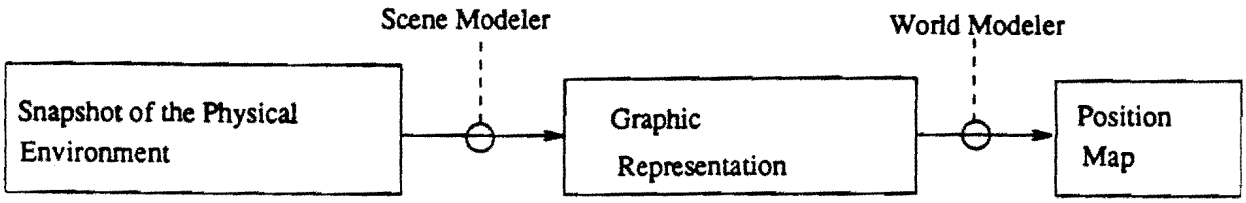


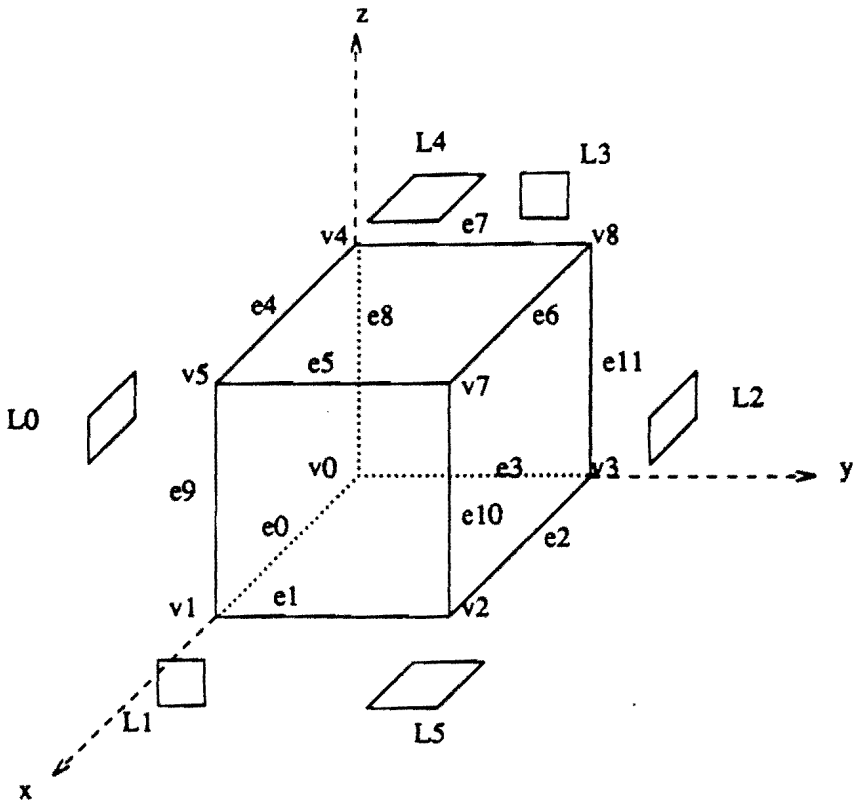
Figure 2.1. Three-Tiered Approach to Obtaining a World Model (Position Map)

Currently, a world model captures a static image of the physical environment. However, objects may move about and interact obeying many principles in the physical environment: collision, conservation of momentum, friction, and stability. The focus of this research is on the interaction between the *collision physics* of the physical environment and the assembly task knowledge used to generate assembly sequences. Although geometric modeling of the shapes of solid objects is well-understood, it is a research problem to model physical principles efficiently. For a geometric simulation of a collision detection, GMS should create an envelope of space that an object may go through when it is moving and check for any intersection with other objects in space, called a sweeping volume. This test becomes almost impossible when other objects are moving as well. Even when all obstacles are stationary, a naive approach would test for intersection of the sweeping volume with all the obstacles. Furthermore, each intersection is very computationally intensive if the shapes are complicated. Here, the collision detection is simulated by the user input: that is, the user may envision whether a moving object collides with other objects using the graphic display of the scene as an aid and informs the reasoner of unintended collisions during an assembly operation.

2.1. Object Shape Modeling

The shape of an object is modeled by its bounding faces. Since a face may have holes, the boundary of a face is formed by its outermost boundary and the inner boundaries formed by the holes: each of them is represented by a sequence of edges, called a loop. Hence, a face is bounded by a loop or multiple loops. Furthermore, a face is associated with an outer normal that points out of the object to distinguish inside from outside of the object. This scheme is called a boundary representation (B-rep) of an object.

A cube shown in Fig. 2.2 (a). A coordinate system is affixed to the cube, called a *base frame*, and the cube is modeled with respect to the base frame. Fig. 2.2. (a) shows the x, y, and z axes of the



(a)

F0 = {L0} F1 = {L1} F2 = {L2}
F3 = {L3} F4 = {L4} F5 = {L5}

(b)

L0 = (e0, e9, e4, e8) L1 = (e1, e10, e5, e9) L2 = (e2, e11, e6,e10)
L3 = (e3, e8, e7, e11) L4 = (e4, e5, e6, e7) L5 = (e0, e1, e2, e3)

(c)

e0 = <v0, v1> e1 = <v1, v2> e2 = <v2, v3> e3 = <v3, v0>
e4 = <v4, v5> e5 = <v5, v6> e6 = <v6, v7> e7 = <v7, v4>
e8 = <v0, v4> e9 = <v1, v5> e10 = <v2, v6> e11 = <v3, v7>

(d)

v0 = <0,0,0> v1 = <1,0,0> v2 = <1,1,0> v3 = <0,1,0>
v4 = <0,0,1> v5 = <1,0,1> v6 = <1,1,1> v7 = <0,1,1>

(e)

Figure 2.2. Boundary Representation of a Cube

base frame.

B-rep of the cube is shown in Fig. 2.2 (b), (c), (d), and (e). A cube consists of six bounding faces: F0, F1, F2, F3, F4, and F5. A face, F0, contains a single loop, L0. F1 contains L1 and so on. These are described in Fig 2.2 (b). A face may contain more than one loop when the face contains holes. A loop consists of edges. L0 consists of four edges: e0, e1, e2, and e3. The rest of the loops with their edges are described in Fig. 2.2 (c). An edge is a straight line segment bounded by a pair of vertices. An edge, e0, is bounded by a pair of vertices, v0 and v1. Fig. 2.2 (d) shows the rest of the edges of the cube. A vertex is modeled by its location, called a *vertex point*. The vertex point is described with respect to the base frame. Fig. 2.2. (e) shows all the vertices and their vertex points with respect to the base frame.

All the geometric information of the boundaries (faces, loops, edges, and vertices) of an object is represented with respect to its base. When an object moves, the base frame affixed to the object is moved accordingly. The geometric information of the boundaries with respect to the base does not change unless the object changes its shape. Since the solid is not allowed to deform by the motion, only the base of an object needs to be updated when the object moves.

Shapes can be altered or constructed by two kinds of operations: resizing and boolean operations. A resizing operation stretches or shrinks an object along the axes of the inertial reference coordinate system of GMS. A boolean operation between a pair of objects creates a new object by subtracting an object from the other, unioning or intersecting them.

2.1.1. Resizing Operation

The x positions of all the vertices of an object are altered proportionately by a scale factor s_x along the x axis of the reference coordinate system of GMS; likewise scaled for their y positions (s_y) and for z positions (s_z). These scaling transformations are carried out by multiplying the following

transformation matrix to each vertex points:

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

No scaling along x axis is necessary by setting s_x to one. Likewise, s_y and s_z can be set to one when scaling along y and z axes are not necessary. The result of resizing operation is stretching or shrinking an object along the axes of the inertial reference frame of GMS. Note, only geometry is affected by this resizing operation but topology of the shape description stays intact. The topology can be altered by the boolean operations.

2.1.2. CSG Boolean Operation

It would be cumbersome for a user to specify all the boundary information for an object model: the vertices, edges, and faces of the cube in Fig 2.2 by hand. When a complicated object is modeled, it would be even more difficult. Therefore, object modeling should make use of similar object models that are already designed in the library. That is, when the user has to specify the boundary information for B-rep of an object, efforts in designing one object model is not easily brought to bear when designing other object models with minor alterations. The user will benefit greatly from tools enabling the use of old object models in building new object models.

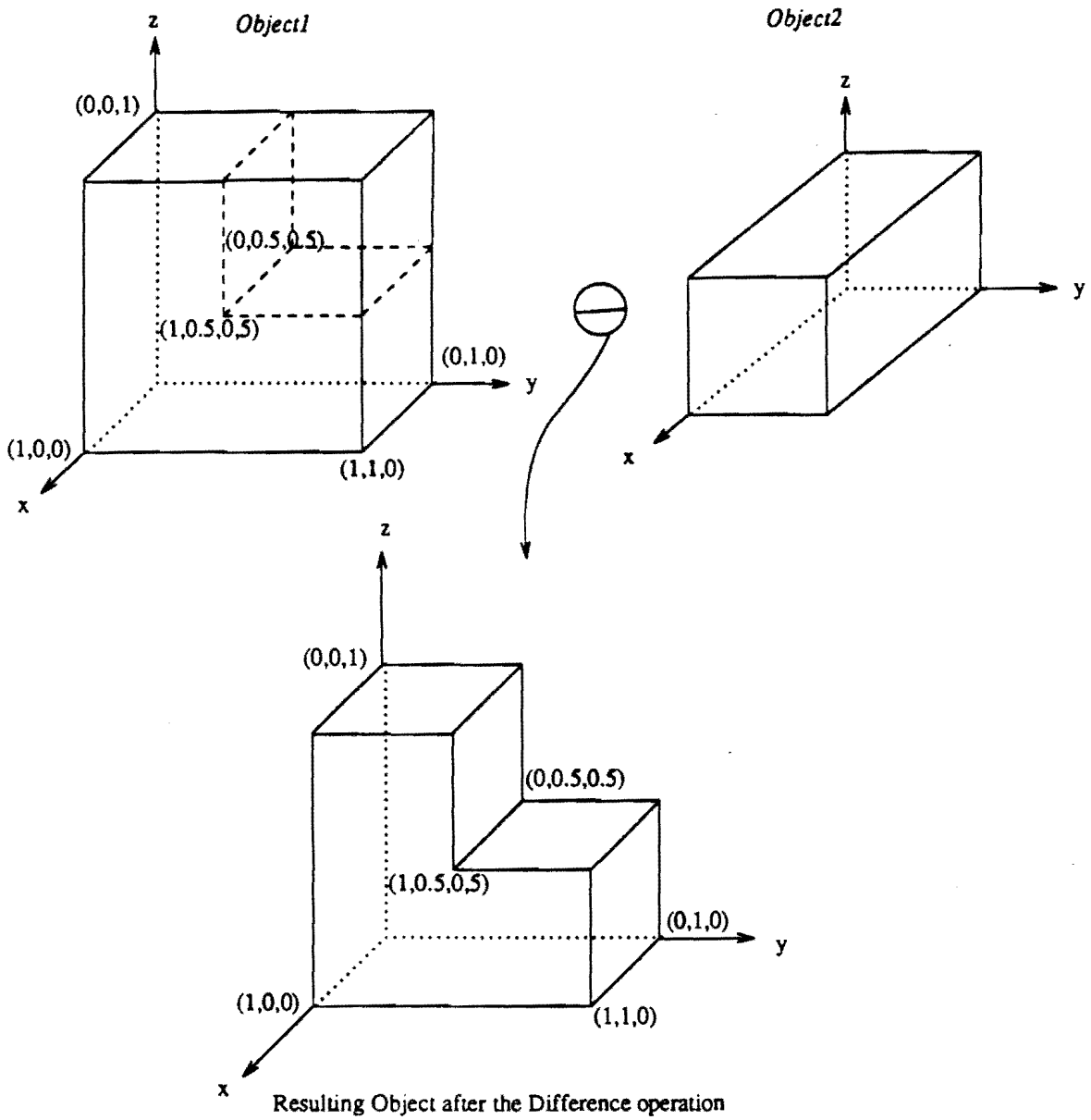
Boolean operators are used to construct union, difference, and intersection of different object models that may be primitives like the cube of Fig 2.2 or previously modeled objects, maybe using boolean operations. Hence, the output object representation of the boolean operation should be uniformly represented as the input object representation: B-rep. This user-interface originates from the constructive solid geometry (CSG) [Requicha & Voelcker 1983]. As a result, an object created by one user may be used or incrementally modified by others or himself at a later time: a design effort is

shared and incremental.

One implementation of the CSG approach forms a binary tree where internal nodes are boolean operators and leaf nodes are primitives [Mortesson 1985]. This computational structure implicitly determines the shape of an object; this implicit object representation is time consuming when the computations of relevant information is duplicated across multiple renderings of objects and geometric reasoning with them. However, B-rep of an object explicitly models the object so that most of the computations are compiled into the representation when the object is created. As a result, B-rep is more efficient for multiple renderings and geometric reasoning.

Here, the GMS evaluates the user commands in terms of CSG boolean operations and creates B-rep of the resulting object. GMS computes all the boundary information of a new object from the boundary information of previously built objects. A boolean operator takes the B-rep of two objects as input. A sequence of boolean operators can be composed because both the input and output objects of the operators are in the same representation, B-rep. Furthermore, the command sequence acts like a schema that can be evaluated to create many different instances with identical shapes.

Currently, a cube is the only primitive object model of GMS but additional primitives can be defined without any change to the modeler itself: its implementation is independent of any particulars of the cube. For example, it would be straightforward to increase the types of primitives even further with pyramid, tetrahedron or any polygonal objects. However, primitives with curved surfaces like a sphere are not directly modeled because GMS is not equipped with solving non-linear equations. This limitation can be overcome by linear approximation of curved surfaces: e.g., a cylinder may be approximated by many-sided polygon.



Resulting Object after the Difference operation

Figure 2.3. Difference Operation of Object2 from Object1

2.1.3. Steps to Object Modeling by a Boolean Operation

Here, the user interface makes use of both boolean and resizing operators. In Fig. 2.3, object1 is a primitive, cube of a unit size; object2 is an elongated rectangular polygon along the x axis of its base by a resizing operator. Then, the base of object2 is placed with respect to object1's base as the inertial reference frame so that object2 is overlaid at the volume enclosed by the dashed lines and the boundary edges of object1. Then, input these two objects to the difference operator that forms new vertices, edges, loops, and faces for the resulting object. The preliminary steps (resizing and placing the objects in common reference frame) are identical to all the other boolean operators.

2.2. Position and Displacement Modeling

A position in 3D space is specified with respect to some coordinate frame. There are many types of coordinate systems but here, the coordinate frames are represented with respect to a right-handed cartesian coordinate system (a *frame*, for short), shown in Fig. 2.4: x, y, and z are the axes and o is the origin. The location of a point in the 3D world, p in Fig. 2.4, is specified with respect to

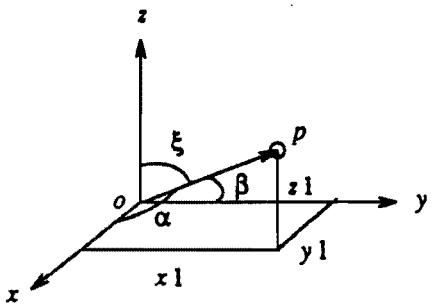


Figure 2.4. Coordinate Frame

the frame by the displacements along the x, y, and z axes of the frame from its origin: x l, y l, and z l. An orientation is specified by a normal vector (a unit segment of the arrow from o to p) with respect to the frame: the cosines of the angle between the vector and the x axis (α), the vector and the y axis (β), and the vector and the z axis (ξ) in Fig. 2.4.

The position of a solid object is specified by the location of a point on the object (the origin of the base frame affixed to the object by convention) and the orientation of the object (the orientation of any of x, y, and z axes of the base frame[1]). Therefore, a position is specified by six parameters: three of the location and three for the orientation. These parameters are called six *degrees of freedom* (DOF) of the position of an object because if any of them is undetermined, the position of the object with respect to the undermined parameter and object is free to move with changing the parameter.

A displacement operator moves an object by a translation, a rotation, or both. The displacement is specified in terms of the six parameters. A mating operation between a pair of components involves the displacement of one component with respect to another such that the two components satisfy their mating conditions in the target assembly.

2.2.1. Position Representation via Homogeneous Transformation Matrix

The translation parameters for the base frame (A) with respect to some coordinate system (B), represented by the origin point of A frame with respect to B: $<p_x, p_y, p_z, 1>^T$ or $\vec{p}$. The rotation parameters are represented by x, y, and z axes of the A frame: $<n_x, n_y, n_z, 0>^T$ or $\vec{n}$ for x axis of A with respect to B, $<o_x, o_y, o_z, 0>^T$ or $\vec{o}$ for y axis, and $<a_x, a_y, a_z, 0>^T$ or $\vec{a}$ for z axis. In short, these vectors, $\vec{n}, \vec{o}, \vec{a}, \vec{p}$, collectively describes A with respect to B and form a homogeneous transformation matrix [Paul 1982]:

[1]The other two axes are fixed when the direction of one of the axes is determined

$$H_K^B = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note, this representation is redundant because there are twelve entries but only six degrees of freedom. Each element of matrix H, can be addressed by specifying the entry name. For example, the first row and column element of H is addressed by $n_x(H)$ or n_x when H is obvious from the context. In addition, each column can be addressed by $\vec{n}$, $\vec{o}$, $\vec{a}$, and $\vec{p}$.

2.3. Scene Modeling and its Rendering

Once all the objects are modeled, they are "placed" on the screen by specifying the homogeneous transformation matrices of their base frames with respect to the *world coordinate system* that is fixed throughout the geometric modeling. The objects are displayed by orthographic projection on a plane. First, represent all the edges of the screen with respect to the world coordinate system; then, project them to the plane. Currently, the rendering routines like hidden line removal, lighting, and a perspective projection, have not been implemented because they bear little bearing toward the main concern of the assembly planning of this thesis research. Furthermore, there are many commercially available graphics packages for the graphic rendering purpose. However, the user may view the scene at different angles by specifying different projecting planes as viewing screens.

2.4. World Modeling by the Position Map

The object models and their placements signify the graphic representation of the scenery. The scenery is perceived by transforming the graphic representation into a digraph, called a *position map*. Paul [Paul, 1982] used a similar structure called a transform graph. There, the usage was mainly for deriving the proper transforms for the *theoretical* analysis of the robot manipulators.

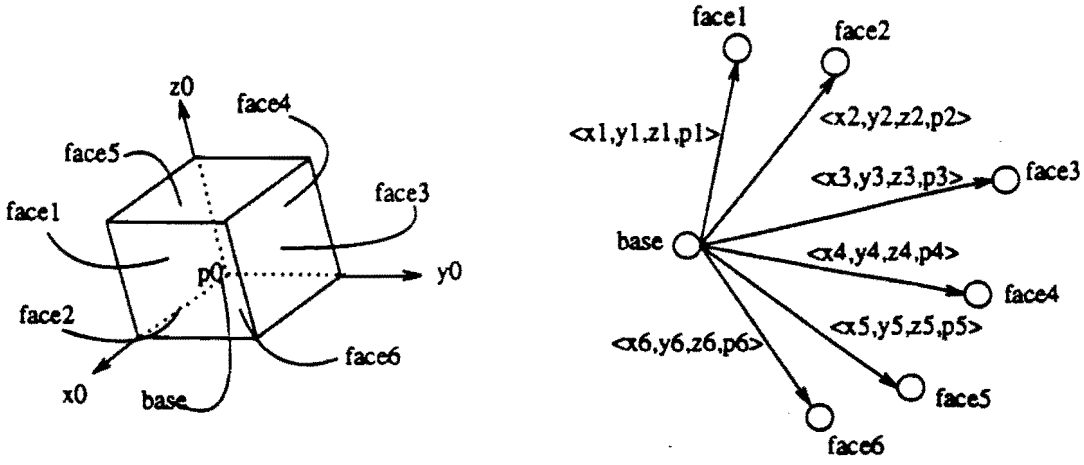
Here, the position map is to represent state of the world. The map includes the digraphs for parts, a node for the world coordinate system (world reference node), and the edges from the world reference node to the base nodes of part digraphs. These edges are labeled by homogeneous transformation matrices signifying the positions of objects in the scene.

2.4.1. Part Digraph in the Position Map

To generate the position map, B-rep of an object is converted into a digraph that contains a base node and nodes for other geometric features: outer normal of a face by a face node and a center axis of a cylindrical hole by an axis node. The edge from the base to a face node is labeled by a homogeneous transformation matrix whose $\vec{a}$ is the outer-normal of the face and $\vec{p}$ lies on the plane of the face. $\vec{n}$ is along one of the edges of the outermost loop of the face, and $\vec{o}$ is chosen following the right-handed coordinate system. The face node contains ranges along $\vec{n}$ and $\vec{o}$ with respect to the frame for the face, approximating the area of the face.

The edge from the base to an axis node is, again, labeled by a homogeneous transformation matrix whose $\vec{p}$ lies on the intersection of the axis and the outer surface; $\vec{a}$ is along the axis pointing inwards from $\vec{p}$. The axis node has a range information along $\vec{a}$ approximating the length of the axis. Because of the convention for choosing $\vec{p}$ and $\vec{a}$, the range is between 0 and some positive number. Note, the range is specified with respect to the frames of each geometric features: the face and axis features.

A simplest external representation of an object is a unit cube. The cube shown in Fig. 2.5 (a) has its own base, and six faces: face1, face2, face3, face4, face5, and face6. So, its internal representation has seven nodes: one for the base and six for the faces. An edge from the reference coordinate system of GMS to the base is labeled by a frame, $<x\vec{0},y\vec{0},z\vec{0},p\vec{0}>$ where $x\vec{0}$ denotes x coordinate axis with respect to the reference coordinate system, $y\vec{0}$ for y coordinate axis, $\vec{z}$ for the z coordinate axis,



External Representation

Internal Representation

(a) B-rep of a cube and its Part Digraph

$$\langle x1,y1,z1,p1\rangle = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix} \quad \langle x2,y2,z2,p2\rangle = \begin{Bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix} \quad \langle x3,y3,z3,p3\rangle = \begin{Bmatrix} -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

$$\langle x4,y4,z4,p4\rangle = \begin{Bmatrix} 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix} \quad \langle x5,y5,z5,p5\rangle = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{Bmatrix} \quad \langle x6,y6,z6,p6\rangle = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

(b) Homogeneous Transformation Matrices Labeling the Edges

Figure 2.5. External and Internal Representation of a Cube

and $p\vec{0}$ for the location of the origin. In Fig. 2.5 (b), an edge from the base to face1, a face node, is

labeled with a frame: $<x\vec{1},y\vec{1},z\vec{1},p\vec{1}>$ where $x1vec$, $y1vec$, $z\vec{1}$, and $p\vec{1}$ are $\vec{n}$, $\vec{o}$, $\vec{a}$, and $\vec{p}$ of face1 respectively; similarly for faces 2 through 6. The origin of its frame, $p\vec{1}$, is at $p\vec{0}$. The ranges along x and y axes of face1 frame are between 0 and 1; here, these ranges exactly describe the boundary. However, in general when the boundary of a face is not rectangular, the ranges only approximate the exact boundary of the face. These ranges are useful when computing the ranges of the degree of freedom variables (they are discussed in the next chapter).

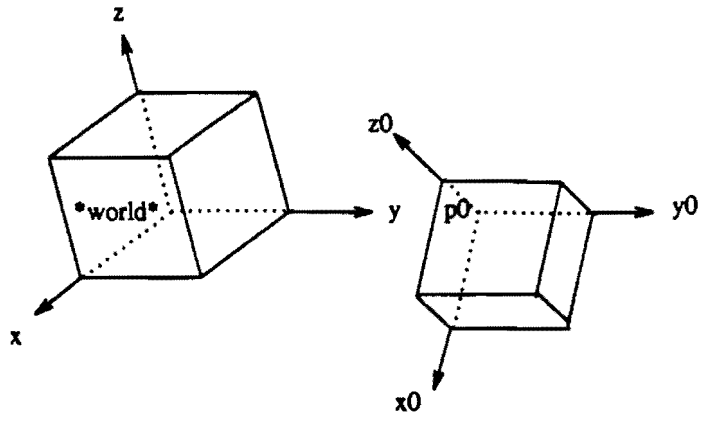
2.4.2. Multiple Scenes with the Position Map

In the position map, a *world* node corresponds to the world coordinate system of the screen. The position of an object in the screen is represented by a matrix labeling a directed edge from *world* node to its base node. The matrix is called a base frame that represents the position of the base affixed to an object with respect to world coordinate system. The scenery is represented by edges from *world* node to base nodes of all the objects in the working space; multiple scenes are maintained by multiple collections of these edges. Neither face frames nor axis frames are changed since objects are assumed to be rigid.

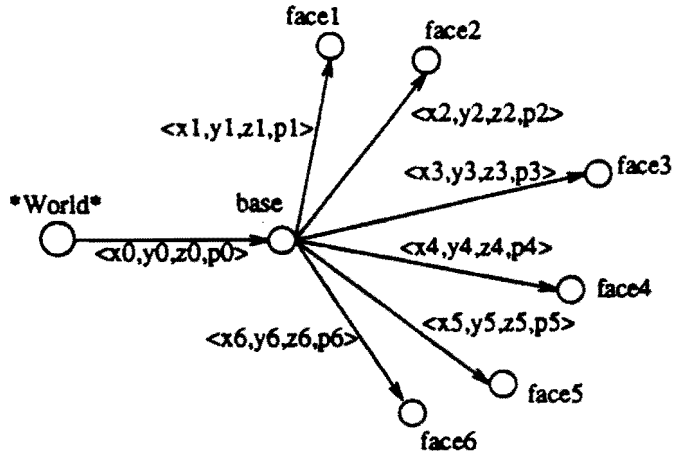
The cube is rotated counter clockwise with respect to y axis of the world reference frame and translated along its y axis by 2. Cubes before and after the motion are shown in Fig. 2.6 (a). $<\vec{x},\vec{y},\vec{z},\vec{p}>$ is the frame for *world* and the original base frame of the cube. The new base frame of the cube after the transformation with respect to *world* is $<x\vec{0},y\vec{0},z\vec{0},p\vec{0}>$:

$$<x\vec{0},y\vec{0},z\vec{0},p\vec{0}> = \begin{bmatrix} 0.707 & 0 & 0.707 & 0 \\ 0 & 1 & 0 & 2 \\ -0.707 & 0 & 0.707 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The position map is labeled with $<x\vec{0},y\vec{0},z\vec{0},p\vec{0}>$ and all the local face frames of the cube with respect to *world* are easy to compute: face1 frame with respect to *world* is computed by matrix-



(a) External Representation of cube before and after transformation



(b) Internal Representation of the Scene
(Position Map)

Figure 2.6. The Scene and the Position Map

multiplying $\langle x\vec{1}, y\vec{1}, z\vec{1}, p\vec{1}\rangle$ with $\langle x\vec{0}, y\vec{0}, z\vec{0}, p\vec{0}\rangle$:

$$Frame\left(\begin{smallmatrix}face1\\world\end{smallmatrix}\right) = \langle x\vec{1}, y\vec{1}, z\vec{1}, p\vec{1}\rangle \times \langle x\vec{0}, y\vec{0}, z\vec{0}, p\vec{0}\rangle = \begin{Bmatrix} 0.707 & 0 & 0.707 & 0 \\ 0 & 1 & 0 & 2 \\ -0.707 & 0 & 0.707 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix} \times \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

$$= \begin{Bmatrix} 0.707 & 0.707 & 0 & 0 \\ 0 & 0 & -1 & 2 \\ -0.707 & 0.707 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

2.4.3. Reasoning with the Position Map

The position map makes it easy to compute spatial relationships between any pair of its nodes: we just have seen how to compute the face frames with respect to *world* when an object moves. In general, to compute a spatial relation of node2 with respect to node1,

(1) find a path from node1 to node2, and

(2) multiply homogeneous transformation matrices of the labels associated with edges of the path appropriately.

A path from node1 to node2 is found by a traversal algorithm: it should treat the directed edges of the position map as undirected edges during the traversal and collects them into a path in a sequence. When a path is found from node1 to node2, collect the proper matrices of the edges in the path: a proper matrix depends on the directed edge. When an edge in the path links node-a to node-b while the traverser went from node-b to node-a, the proper matrix is the inverse of the matrix labeling the edge; otherwise, the label is the proper matrix as it is.

Suppose you want to compute the spatial relation of face1 with respect to face2 in the scene of Fig. 2.6 (b). The labels of the path from face2 to face1 are $\langle x\vec{1}, y\vec{1}, z\vec{1}, p\vec{1}\rangle$ and $\langle x\vec{2}, y\vec{2}, z\vec{2}, p\vec{2}\rangle$.

Because the path requires the edge from face2 to base, inverse of $\langle \vec{x}2, \vec{y}2, \vec{z}2, \vec{p}2 \rangle$ is required:

$$\langle \vec{x}2, \vec{y}2, \vec{z}2, \vec{p}2 \rangle^{-1} = \begin{Bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

Then, face1 with respect to face2, $Frame^{face1}_{face2}$ is a matrix multiplication of $\langle \vec{x}1, \vec{y}1, \vec{z}1, \vec{p}1 \rangle$ and $\langle \vec{x}2, \vec{y}2, \vec{z}2, \vec{p}2 \rangle^{-1}$:

$$Frame^{face1}_{face2} = \langle \vec{x}1, \vec{y}1, \vec{z}1, \vec{p}1 \rangle \times \langle \vec{x}2, \vec{y}2, \vec{z}2, \vec{p}2 \rangle^{-1} = \begin{Bmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

CHAPTER 3.

ASSEMBLY DESIGN

When individual objects are designed, an assembly structure is formed by including the objects as its components and by specifying the connections between the components in the assembly structure. These connections are formed by positioning the objects where they can form a joint: a kinematic connection to be described later in the chapter. Therefore, the main tasks in the assembly design are to design individual components and to specify their connections in the target assembly. The computer aided design (CAD) systems have provided tools for the component design and increasingly more products are designed by CAD systems. However, many of the products are assemblies; hence, CAD systems should provide tools for the assembly design [Lee & Gossard 1985] in addition to those for the component design.

The assembly design creates a *target assembly*: its geometric and topological structure. The geometric structure of the target assembly is created when the geometric structure of its components are designed by the component design system using the boolean operators, as discussed by the previous chapter. Then, the topological structure of the target assembly is represented by a diagram, called **target assembly diagram**: the main subject of this chapter.

Furthermore, the assembly design should be tested for its assemblability [Ko & Lee 1987]. This test requires an assembly planner that can decide whether the assembly is feasible to construct by creating an assembly plan. As a result, the target assembly diagram specifies the tasks to be performed by the assembly planner; this chapter examines the assembly design as a *task specification process* for the assembly planner.

3.1. User Interface for the Assembly Design

Assembly design is carried out by the user with a CAD system. The level of user instructions to the CAD system is the subject of this section.

3.1.1. Coordinate-based Approach

The interfaces in commercial CAD systems are awkward because repositioning of individual components are done by:

- pointing to an object using a mouse, and dragging to its desired joint configuration; then, select a joint type (to be discussed shortly) from a pop-up menu and its associated axes of kinematic degrees of freedom, or
- specifying a homogeneous transformation matrix directly and then select their joint type from a menu and their axes of kinematic degrees of freedom.

The former is easier but lacks accuracy while the latter is more precise but lacks ease of usage. For either method, the user has to specify the translations and rotations necessary for repositioning individual components manually, a time consuming process. It would be better if the kinematic connections can be specified without sacrificing the accuracy.

3.1.2. Feature-based Approach

An assembly is designed to serve some function. With the coordinate-based approach, little of the functional requirements of the assembly design are reflected in the representation of the target assembly. Rather, an assembly design should be expressed in terms of the linguistic features describing the shapes, the spatial relations, and the motions: hence the need for the feature-based approach.

The feature-based approach allows the user to design in terms of features of the shapes and their symbolic relationships capturing the required spatial and kinematic requirements. However, with the feature-based approach, the coordinate information should be *inferred* from the features and their symbolic relationships. The coordinate information is necessary for the graphical display or for the robotic manipulation of the assemblies. In NOMAD, the assembly design adopts the feature-based approach.

3.2. Target Assembly Diagram

The target assembly diagram is a "meta" construct over the position map wherein the nodes denote the base nodes of the position map and the edges, called "kinematic liaisons", denote their *kinematic* linkages in the target assembly. The target assembly diagram is based on the earlier work by Bourjault. He characterizes an assembly as a *liaison diagram* [De Fazio & Whitney, 1988]: a labeled graph wherein nodes represent parts and edges between nodes represent *user-defined* relations between parts, called "liaisons".

The liaison diagram allows the user to assign any relational meaning between parts to a liaison while the kinematic liaison is limited to only kinematic relations. Furthermore, two components, connected by a kinematic liaison, should form physical contacts to serve some mechanical purposes. Furthermore, nodes in the target assembly diagram are base nodes of the position map rather than parts. In short, the target assembly diagram can be viewed as a specialization of the liaison diagram.

3.2.1. Related Concepts to Kinematic Liaisons

3.2.1.1. Virtual Link

The kinematic liaisons of the target assembly diagram are closely related to the concept of a virtual link [Lee & Gossard, 1985]. A virtual link denotes a connection between two mating structures: two subassemblies, two components, or one subassembly and one component. Because a subassembly is a hierarchical entity, a virtual link involving subassemblies is defined in the context of a particular assembly hierarchy.

In comparison, a kinematic liaison is specified independent of any particular assembly hierarchy. Instead, the assembly hierarchy is constructed during the planning phase rather than during the design phase. As a result, the assembly hierarchy construction is independent of the assembly design processes. However, the assembly design considerations are pertinent to the assembly hierarchy construction. For example, an engine assembly is designed as a subassembly of the car. The engine subassembly represents a functional unit as a power plant of the car as well as a structural unit that can be assembled or disassembled collectively.

These distinctions stem from the property of the hierarchical representation to facilitate an abstraction mechanism. Because different types of abstractions are required by the assembly design and planning phases, separate hierarchies should be constructed for each type of abstractions: functional hierarchy for the former and structural hierarchy for the latter. Here, the assembly hierarchy is constructed only during the assembly planning phase (the type of abstraction represented by the assembly hierarchy will be discussed in Chapters 5 and 6).

Lastly, the virtual link allows both a kinematic and a dynamic linkage information such as a

conditional attachement[1] [Wesley & Lozano-Perez & Lieberman & Lavin & Grossman 1980].

3.2.1.2. Joint (Lower-Pair)

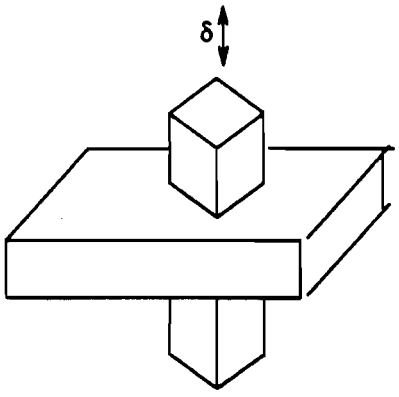
A joint between two geometric features, each from different objects denotes a kinematic constraint between them. Their relative spatial movements are constrained geometrically as a function of six parameters denoting three translational degrees of freedom and three rotational degrees of freedom. The geometric features related by a joint is called *elements*. A joint and its pair of of elements is called a *pair*. In 1875 [Kennedy, 1876], Reuleuax distinguished the joints into six types: prismatic, cylindrical, spherical, revolute, screw, or planar joint; a pair from these six joint types is called a lower-pair [Kenedy 1876; Hartenberg & Denevit 1964]. Their kinematic properties are shown in the Fig. 3.1.

The kinematic properties of these joint types has been made precise by a homogeneous transformation matrix [Denevit & Hartenberg, 1955], modeling the motions of the lower-pair joints. The matrix has historically been called an **A matrix**. The main application of their approach has been the mechanism simulation of an assembly structure when parts are already in place.

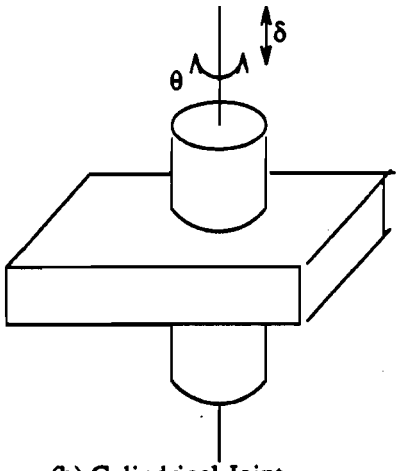
Because their approach assumes all the parts are in their assembled positions already, there are no displacements necessary to bring about the joint. However, for the assembly construction task, components may be initially at arbitrary positions and they should be brought together in space to form an assembly. Hence, their displacements as well as modeling the mechanism behavior are necessary (the subject of the next chapter).

The kinematic liaison represents the overall kinematic constraint between base nodes of two components while a joint represents a kinematic constraint between their local geometric features. Then, a kinematic liaison may represent the resulting kinematic constraint from multiple lower-pairs.

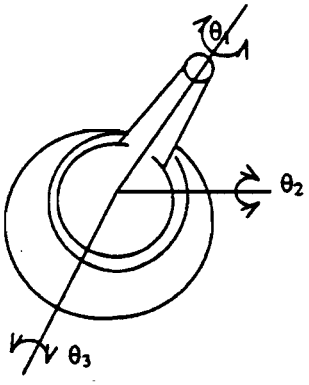
[1] A conditional attachment may have a relative motion if the applied force is greater than some threshold.



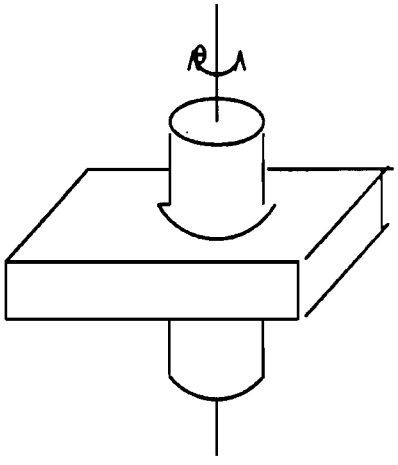
(a) Prismatic Joint



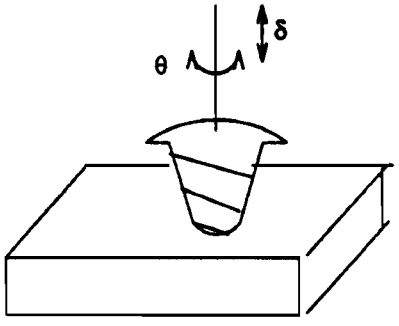
(b) Cylindrical Joint



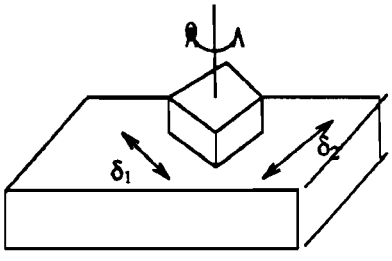
(c) Spherical Joint



(d) Revolute Joint



(e) Screw Joint



(f) Planar Joint

Figure 3.1. Six Lower Pairs and their Kinematic Properties

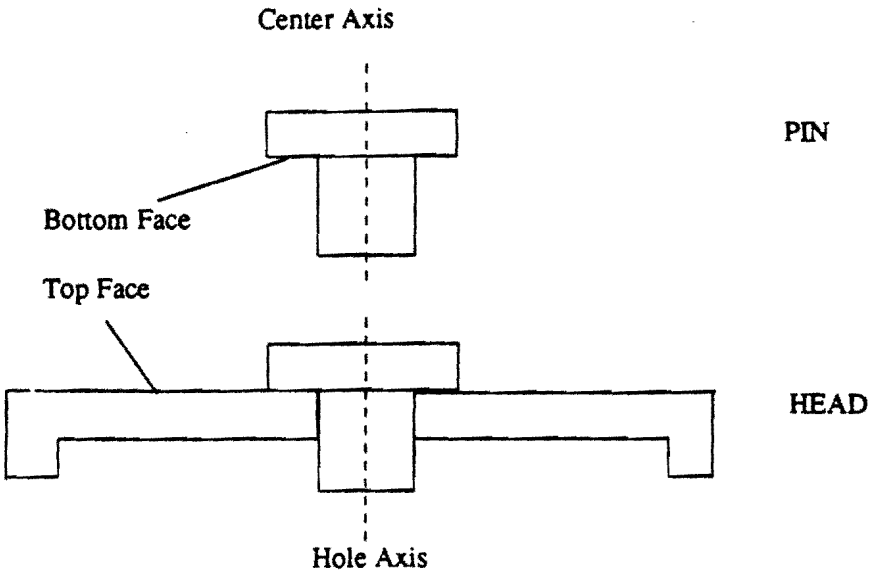


Figure 3.2. Elements of PIN and HEAD and their Joint Formation

For example, in Fig. 3.2, when the PIN is lowered, its bottom face forms a planar joint with the top face of the HEAD. Furthermore, the center axis of the HEAD forms a cylindrical joint with the hole axis of the HEAD. Overall, the HEAD and PIN form a revolute joint: that is, the kinematic liaison between base nodes of HEAD and PIN represents a revolute joint, consisting of planar and cylindrical joints.

3.2.1.3. Mating Condition

These individual local kinematic constraints are related to the *mating conditions* [Popplestone & Ambler & Bellos 1980]. A mating condition denotes both the joint to be established and the necessary displacements. For example, the planar and cylindrical joints are established by satisfying different types of mating conditions: against and aligned mating conditions (to be discussed shortly). In

short, a kinematic liaison may contain more than one mating conditions. Furthermore, a mating condition denotes not only the final joint properties but also the displacements required to achieve the joint. That is, a mating condition is a function of both the initial configuration of parts and their joint configuration.

3.2.2. Specifying a Mating Condition

NOMAD allows the user to specify the kinematic constraints of the kinematic liaison *indirectly* in terms of the mating conditions. Each mating condition specifies the transformation from the current configuration to the joint configuration between local geometric features. When the joint allows the kinematic degrees of freedom, different transformations are allowed as long as they differ only by the kinematic degrees of freedom of the joint. However, the transformation should not be too wild: the elements of the joint should keep a physical contact. Here, ranges of the degree of freedom variables are restricted to satisfy this constraint (to be discussed in Chapter 4).

In order to specify a mating condition, the user specifies the pair of geometric features (elements of the joint) and the type of the joint that they form: a mating condition for each of the six joint types in Fig. 3.1. There could be other mating conditions for additional joint types however, the completeness of the mating conditions should be tested with many assembly examples.

As an example, a mating condition between two planar faces to form a planar joint may be specified and is called an *against mating condition*. Another example is a mating condition between two axes to form a cylindrical joint, called as an *aligned mating condition*. The assembly of the PIN and HEAD in Fig. 3.2 is specified by an against mating condition between the bottom and top faces and an aligned condition between the center and hole axes. Chapter 4 develops a method that derives the overall kinematic constraint, forming a revolute joint, between PIN and HEAD from each individual against and aligned mating conditions. Next two sections describe the mating conditions in the

context of the position map framework.

3.2.2.1. Against Mating Condition

Suppose f1 and f2 of Fig. 3.3 are two planar face nodes. The frames for f1 and f2 with respect to some common reference coordinate system are $<\vec{u},\vec{v},\vec{w},\vec{p}>$ for f1, and $<\vec{x},\vec{y},\vec{z},\vec{o}>$ for f2. Remember $\vec{w}$ and $\vec{z}$ are outer normals of f1 and f2 respectively; $\vec{p}$ and $\vec{o}$ lie on the planes of f1 and f2.

Informerly, two planar faces are *against* each other when they lie in the same plane and face each other: that is, $\vec{w}$ and $\vec{z}$ are antiparallel and $\vec{p}$ and $\vec{o}$ lie in the same plane. In addition, two planar faces form a planar joint: the planar joint allows three degrees of freedom: one rotational degree of freedom around the outer normals when they are antiparallel and two translational degrees of freedom on the plane shared by the two faces. The degree of freedom variables are associated with a range

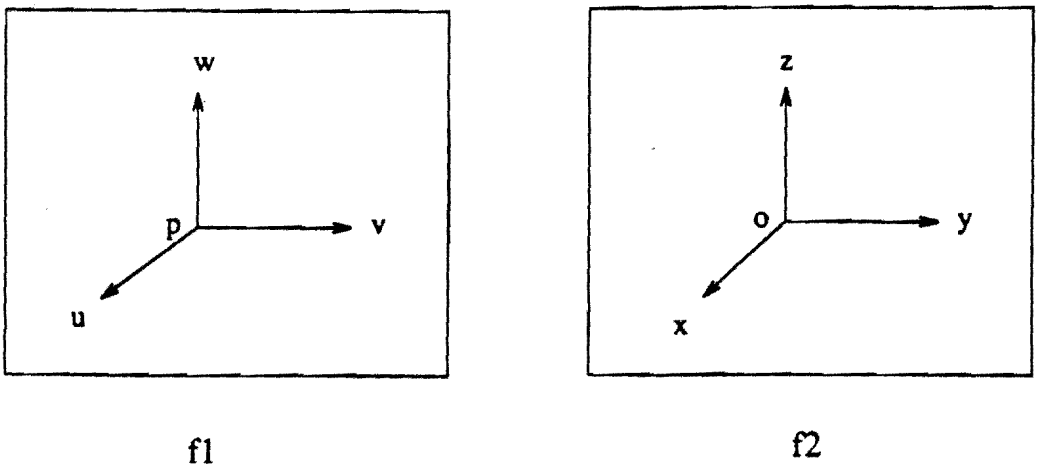


Figure 3.3. Object Features (Kinematic Elements)

restriction so that two faces form a physical contact. However, this is in general a function of all three degrees of freedom variables and edges of the loops in the faces. NOMAD approximates the restrictions by the translational degrees of freedom along $\vec{u}$ and $\vec{v}$ of f1 if the object of f2 is moved for their mating using the range information of f1 and f2. Chapter 5 details the mechanism using an example.

3.2.2.2. Aligned Mating Condition

Here, suppose f1 and f2 are the axes nodes. An example would be a center axis of a hole, the hole axis in Fig. 3.2. Again, the frames for f1 and f2 with respect to some common reference coordinate system are $<\vec{u},\vec{v},\vec{w},\vec{p}>$ for f1, and $<\vec{x},\vec{y},\vec{z},\vec{o}>$ for f2. Remember $\vec{w}$ and $\vec{z}$ are along the axes of f1 and f2 respectively; $\vec{p}$ lies on the intersection of the axis and the outer surface. Informerly, two axes are *aligned* each other when they are collinear and coincident. More precisely, when f1 and f2 are aligned, $\vec{w}$ and $\vec{z}$ are collinear and $\vec{p}$ and $\vec{o}$ are lying along a common axis. In addition, they form a cylindrical joint with two degrees of freedom: one rotational degree of freedom around the common axis, and one translational degree of freedom along the common axis. The range information can be derived similarly.

3.2.3. Example of Target Assembly Diagram

As an example, Fig. 3.5. shows two components: "Bench" on the left and "Beam" on the right. Bench constains faces, f1 and f2; Beam constains faces, f3 and f4. The assembly of the Bench and Beam is specified by two mating conditions: Faces, f1 and f3, are against each other and faces ,f2 and f4, are against each other. State, S1, shows before their mating conditions are satisfied and S2 shows afterwards.

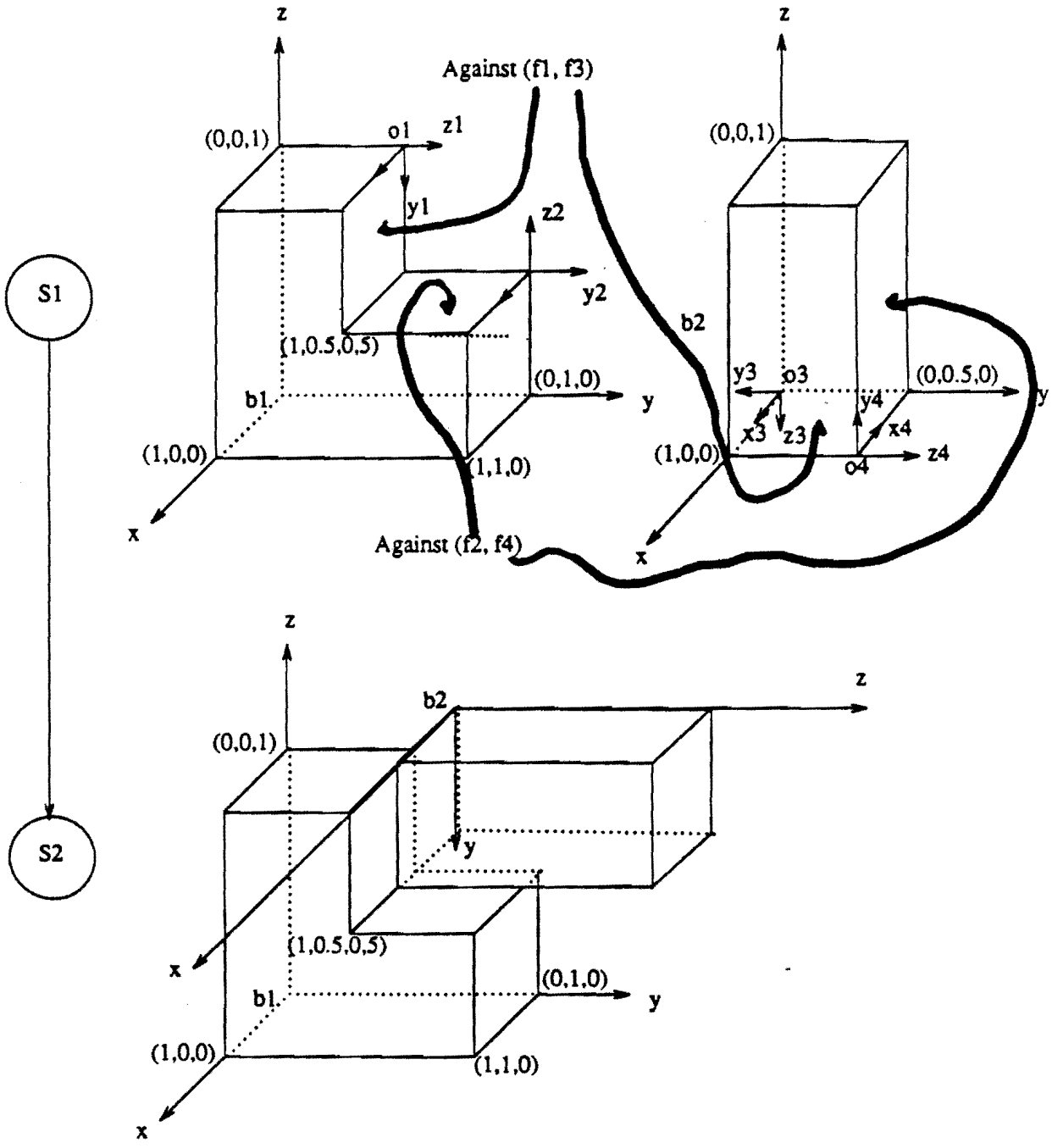


Figure 3.5. Bench Assembly Specification of its Assembled State

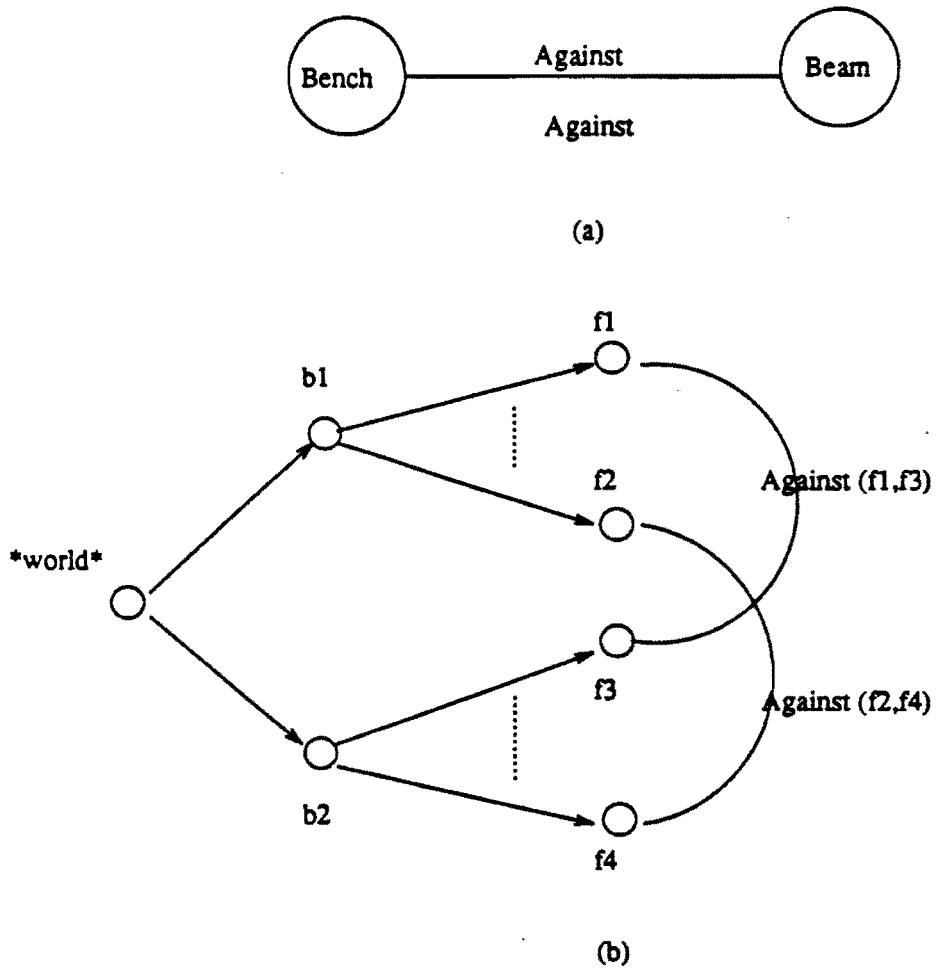


Figure 3.6. Target Assembly Diagram for Bench Assembly

Fig. 3.6 (a) shows the target assembly diagram on the abstract where nodes are base nodes of Bench and Beam, b1 and b2. Here, the names of components themselves are used to denote b1 and b2. The kinematic liaison is labeled by two against mating conditions between them. Fig. 3.6 (b) shows the detailed representation at the target assembly diagram overlaying the position map for Bench and Beam.

CHAPTER 4.

SPATIAL REASONING ENGINE

The spatial reasoning engine (SRE) is a set of inference procedures, reasoning about the geometric and the topological relationships between the geometric features of the objects in the position map. SRE can be used for computing a new position of a component after mating with another component or for envisioning a subassembly after all the mating conditions of its components are satisfied, resulting in the final joint with kinematic[1] degrees of freedom (DOF).

Given one or more mating conditions between a pair of components, SRE computes the displacements of one component to the other component, satisfying their mating conditions simultaneously.

Given:

- Current position map of the pair of components: their geometric features and their current positions.
- A set of mating conditions to be satisfied after the assembly operation.

Determine:

- Displacements including the translations and rotations that would achieve the given mating conditions.

The DOF is represented by variables, called DOF variables. When the final joint allows DOF, there are infinitely many possible displacements satisfying the final joint condition. Therefore, all the valid displacements (a group of rigid transformations) are represented by a homogeneous transformation matrix whose entries may contain the symbolic constraint equations in terms of the DOF variables. Furthermore, after displacements, all the mated geometric features should be physically in

[1]Ampere coined the term, kinematics, derived from the Greek word for motion, cinematique: a geometric science of motion [Hartenberg & Denevit 1964].

contact: they should intersect in space. This is managed by the range restrictions on the DOF variables.

In order to execute the mating operation, the kinematic DOF variables should be grounded: that is, the variable should be instantiated to a numerical value so that the position of the object can be fully determined so that the scene can be displayed graphically.

The DOF variable should be grounded within its specified range to ensure the contacts. The grounding within the range determined locally between two mating components may be an overcommitment when the grounded value is later determined to be unacceptable because a placement of an object using the grounded value hinders later mating operations. Therefore, the satisfaction of other mating conditions in the target assembly may further constrain the choice of numerical values for grounding a DOF variable.

At worst, considering the mating conditions of an assembly in their entirety may be necessary to minimize backtracking from a grounded value. The backtracking alters the placement of a component and may destroy the placements of other components in the assembly structure built so far. In short, SRE should be capable of determining spatial relationships from any subset of all the mating conditions, collected from the kinematic liaisons in the target assembly diagram.

4.1. Previous Researches to the Inference of Positions

4.1.1. Algebraic Approach

Inference of positions from mating conditions was originated from intelligent robotics [Popplestone & Ambler & Belos, 1980]. Their objective was to elevate the manipulator-level instructions to the level of specifying only spatial-relationships between objects. This requires deriving

manipulator-level instructions automatically from the spatial-relation specification.

The objects were modeled by the collection of geometric features participating in the spatial relationships rather than by a geometric modeling system. The derivation of the displacements were carried out by an algebraic system, modeled by a set of rewrite rules with the following operators: twix, trans, neg, vec, cos, sin. The inference of positions from the mating conditions is carried out by solving the equations *algebraically* and contributed to the apparent inefficiency of the method. The concept of twix and trans were incorporated in SRE.

4.1.2. Numerical Approach

More recently, it was realized that inference of positions is necessary to support an assembly design in the CAD environment [Lee & Gossard 1985; Lee & Andrew 1985]. The term, "mating condition", originated from their work.

The necessary displacements were derived by solving a set of simultaneous constraint equations derived from each mating conditions numerically. Some of these equations involve trigonometric functions, modeling the rotations. As a result, their method involved solving a non-linear set of equations.

All the twelve entries of a homogeneous matrix are considered as independent variables and their constraint equations were explicitly posted as part of the simultaneous set of equations. There was more equations and unknowns than minimal number to compute the spatial displacements. Furthermore, there were usually more equations than the number of unknowns. It was necessary to isolate the same number of *independent* equations as the number of unknowns: a type of reasoning over the form of equations.

In order to solve the non-linear equations, it was necessary to guess the initial values of the unknowns so that the numerical solution method converges. Because initial values determine the initial configurations of the components, the designer supplied the initial configurations of the parts in the assembly. As a result, the designer was not completely freed from specifying the positions of the components: this is unavoidable when the constraint equations involve non-linear terms.

SRE extends the above method in three ways. First, computing displacements for each mating conditions is compiled into the algorithm using a vector algebra instead of posting the same set of equations everytime a mating condition is used to derive the displacements. Second, it layers the computation according to the degree of difficulties from simple matching to linear equation solving by Gaussian elimination method, and eventually to solving non-linear set of equations. Note, if a simple matching is sufficient to solve the constraint equations, then the numerical method is not used at all. Third, each DOF variable is restricted to an interval so that the range of displacements can be derived for the contact constraint.

4.2. Spatial Reasoning from Mating Conditions

SRE first computes a homogeneous transformation matrix, satisfying the geometric definition of the mating condition, given in the previous chapter. For multiple mating conditions, the matrices for each mating condition are compared and simplified using a symbolic equation simplifier and a *constant dominance* rule (to be explained shortly). Then, its DOF variables are constrained to achieve the contact. That is, the DOF variables of the resulting joint are restricted by an intersection of the ranges of the individual DOF variables from each mating conditions.

The following sections describe an algorithm, deriving the homogeneous transformation matrix from an against mating condition under the position map framework. In the interest of clarity, the position map is simplified: the base nodes of the mating components and the *world* node of a

position map were simplified to a single node. In the simplified position map, an edge links a feature node to a common reference coordinate system marked by "I". Furthermore, in order to simplify duplicating the whole position maps at different stages of the operation of the algorithm, a geometric feature at different stages is represented by attaching a subscript: e.g., $f1_i$ denotes the feature, $f1$, at the ith stage.

4.2.1. Decoupling Kinematic Constraints of Against Mating Condition

The geometric definition of an against mating condition is given in the previous chapter. Note, the definition of the against mating condition is divided into *rotational* and *translational* components: the rotational component specifies $\vec{r}$ and $\vec{w}$ to be antiparallel and the translational component specifies $\vec{p}$ and $\vec{o}$ to be coplanar, after achieving the against mating condition.

It is known that a transformation matrix that would rotate around a unit vector, $\vec{k}$, by an angle, θ, is:

$$Rot_{k,\theta} = \begin{bmatrix} k_x^2 v_\theta + c_\theta & k_x k_y v_\theta - k_z s_\theta & k_x k_z v_\theta + k_y s_\theta & 0 \\ k_x k_y v_\theta + k_z s_\theta & k_y^2 v_\theta + c_\theta & k_y k_z v_\theta - k_x s_\theta & 0 \\ k_x k_z v_\theta - k_y s_\theta & k_y k_z v_\theta + k_x s_\theta & k_z^2 v_\theta + c_\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where c_θ and s_θ are $\cos(\theta)$ and $\sin(\theta)$ respectively and $v_\theta = 1 - c_\theta$ [Paul 1982].

Using the above transformation matrix, the $\vec{r}$ and $\vec{w}$ are made antiparallel by deriving the $\vec{k}$ that is orthogonal to the plane formed by $\vec{r}$ and $\vec{w}$. Rotate $\vec{r}$ around $\vec{k}$ by the angle θ to make $\vec{r}$ anti-parallel to $\vec{w}$.

$$\theta = 180 - \text{acos}\,(\vec{r} \cdot \vec{w})$$

$$\vec{k} = |\ \vec{w} \times \vec{r}\ |$$

$$RotFramef2 = Rot_{k,\theta}\, Frame2f2$$

where $|\ \vec{w} \times \vec{r}\ |$ is a unit vector and $RotFramef2$ is a new frame of $f2$ after rotation. In Fig. 4.1, $f2_i$

represents the feature $f2$ after $RotFrame/^2$ transformation.

The translational component is derived by noting that once $f2$ is oriented antiparallel to $f1$, the origin of $f2$ with respect to current frame of $f1$ should be reduced to 0 along its current z axis. Offset represents $f2$ with respect to $f1$ after orientation.

$$OffSetf_2^? = Frame\,1f_1\ RotFrame/^2$$

Note that $Frame\,1f_1$ and $Frame\,1/^1$ are inverse to each other. Inverse of a homogeneous transformation matrix requires little computation. Inverse of H in Chapter 2 is:

$$H^{-1} = \begin{bmatrix} n_x & n_y & n_z & -\vec{p}\cdot\vec{n} \\ o_x & o_y & o_z & -\vec{p}\cdot\vec{o} \\ a_x & a_y & a_z & -\vec{p}\cdot\vec{a} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let the z component of translation vector, $p_z(OffSet)$, is d. Let us denote a translational matrix with three degrees of freedom (p_x, p_y ,and p_z) as $\mathrm{Tr}(p_x,p_y,p_z)$.

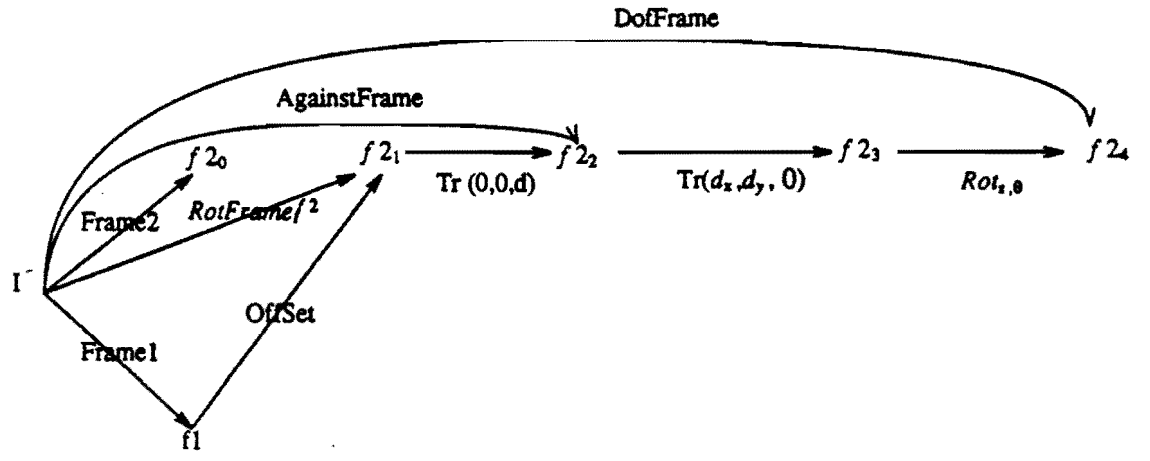


Figure 4.1. Position Map

$$Tr(p_x, p_y, p_z) = \begin{Bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

So, the new frame of $f2$ that satisfies against mating condition with $f1$ is:

$$AgainstFrame/^2 = RotFrame/^2\, Tr(0,0,d)$$

4.2.2. Degrees of Freedom for Mating Conditions

The kinematic degrees of freedom are preserved by allowing numerical equations with variables in the matrix. As a result, symbolic equations should be manipulated to infer spatial relationships. For example, *against* mating condition allows one rotational degree of freedom and two translational degrees of freedom. Here, $f2_2$ may rotate around its current z axis by θ and translate in x-y plane by d_x and d_y. The new transformation matrix with DOF variables (θ, d_x, d_y) for the *against* mating condition is derived by matrix multiplication:

$$DofFrame/^2 = AgainstFrame/^2\, Tr(d_x, d_y, 0)\, Rot_{z,\theta}$$

4.2.3. Reasoning about Ranges

When two geometric features mate by an against mating condition, they should be in contact physically: i.e., their planes should intersect in space. So, the above transformation alone is still underconstraining because depending on DOF variables, the two planes may not touch each other. The spatial contacts are enforced by the range constraints of the DOF variables of the against mating conditions: θ, d_x, and d_y.

Unfortunately, the range of one DOF variable may be a function of other DOF variables. For example, the two unit squares of Fig. 4.2 should be against each other. Then, the two squares mate by a translation of square2 along x axis, d_x, between -3 and -1 with its current orientation. When you

reorient square2 by θ, shown in Fig. 4.2(b), the range constraints on d_x is a function of θ where $0 \le \theta \le 180$.

$$-2-\cos(\theta) \le d_x \le \cos(90-\theta)-1.$$

Hence, translational degrees of freedom in against mating condition between two planes are functions of the rotational DOF, θ, and their boundaries; a notable exception is circular planes because the shape is invariant with respect to any orientation.

In general, each bounding edges from the mating planes should be tested for intersection when the shape of the plane irregular. In this work, we envelope the plane by a rectangle: ranges along the x and y axes of the face frame, affixed to the plane. For an against mating condition, the displacements necessary to achieve spatial contacts are derived from the respective envelopes and the rotational dependency is ignored. Therefore, the displacements may not ensure spatial contacts; instead, it approximates the nearby regions. A complete geometric solution may not be necessary when the

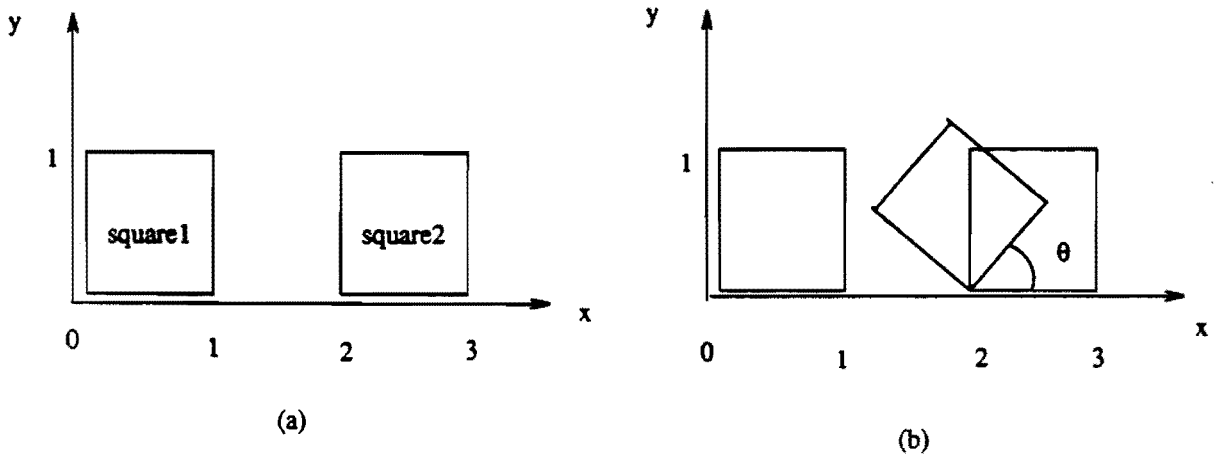


Figure 4.2. Mating Square2 against Square1

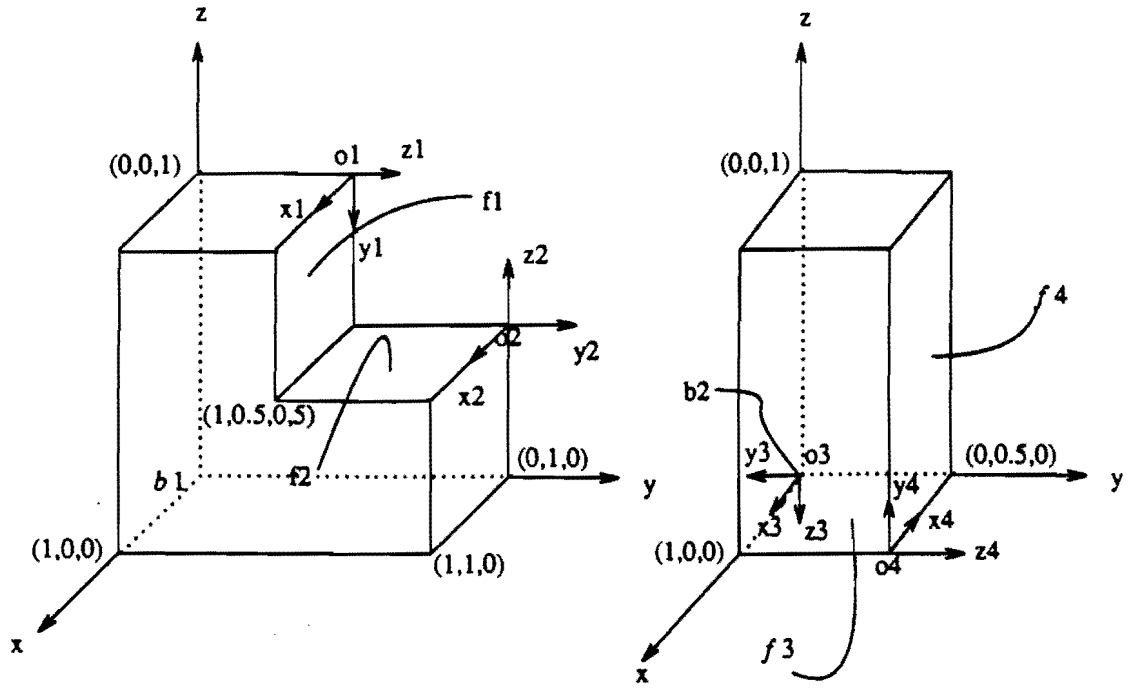
geometric constraint is augmented with the force sensors that can "feel" the contacts through an exploration of the nearby space in small increments: similar to the compliant motion in the robot control [Mason 1979].

4.2.4. Case Study: a Bench Assembly

Using the Bench assembly as an example, Fig. 4.3 (a) shows Bench is the component on the left and Beam is positioned to the right. The dimension of the Beam is 1 by 0.5 by 1 along x, y and z axes respectively. Three geometric features are shown in the figure for each component, one base and two face features. Remember, in the position map, the frame of the base, base frame, of an object is represented with respect to the world inertial reference frame, *world*, and all the other features of the object are defined with respect to its base. Here, $f1$ and $f2$ are face features whose frames are defined with respect to the base, $b1$, of the Bench and $f3$ and $f4$ with respect to the base of the Beam, $b2$: their homogeneous transforms are given in Fig. 4.3(b). Suppose the corner of the Beam between $f3$ and $f4$ is fit into the groove of the Bench between $f1$ and $f2$. Then, two against mating conditions should be satisfied between $f1$ and $f3$ and between $f2$ and $f4$. *DofFrame* 3 is derived as the new base frame of $b2$ with respect to BaseFrame2 after mating $f1$ and $f3$.

$$DofFrame\,3 = \begin{Bmatrix} c_{\theta_1} & s_{\theta_1} & 0 & d_1 \\ 0 & 0 & 1 & -1.5 \\ s_{\theta_1} & -c_{\theta_1} & 0 & d_2-2 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

Here, θ_1, d_1, and d_2 are the rotational DOF variable and two translational DOF variables of the against mating condition between $f1$ and $f3$. *DofFrame* 3 is with respect to the BaseFrame2. Here, the new translational DOF variable is along the x and z axes with respect to the initial base frame of the Beam in Fig. 4.3 (*BaseFrame* 2); notice that the translational component with respect to the y axis of BaseFrame2 is fixed to -1.5. This can be checked intuitively because when the Beam is reoriented so that $f1$ and $f3$ are against to each other, the translation along y direction of the base frame of the



(a) Geometric Models of Bench (at left) and a Beam (at right)

$$Frame\ 1{}^{1}_{f1} = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.5 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

$$Frame\ 2{}^{2}_{f1} = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

$$Frame\ 3{}^{3}_{f2} = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

$$Frame\ 4{}^{4}_{f2} = \begin{Bmatrix} -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

$$BaseFrame\ 1{}^{1}_{world} = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

$$BaseFrame\ 2{}^{2}_{world} = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

(b) Geometric Features

Figure 4.3. Bench and a Beam

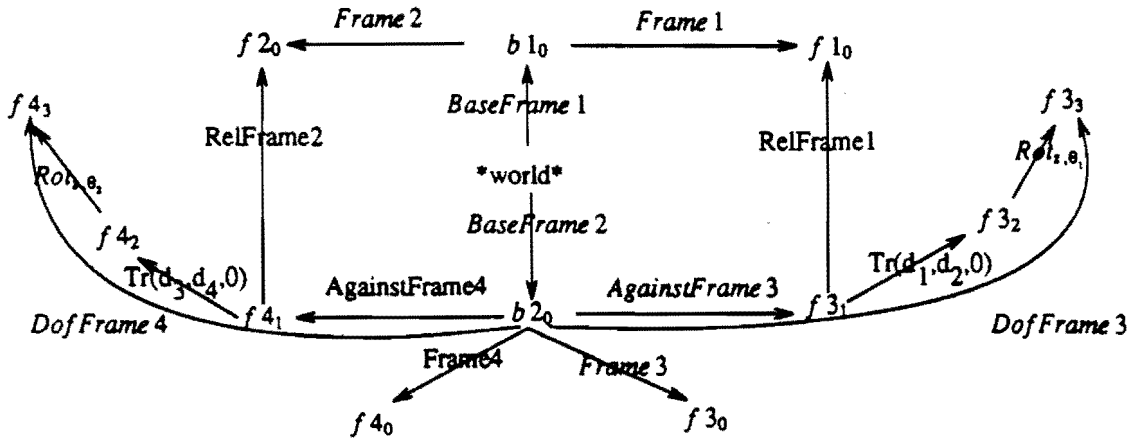


Figure 4.4. Computing the Spatial Relationships of the Beam
and the Bench from Two Against Mating Conditions in the Position Map.

Beam is the only one that is fixed. Similarly for $f2$ and $f4$, the new base frame of the Beam with respect to *BaseFrame* 2 is *DofFrame* 4.

$$DofFrame4 = \begin{Bmatrix} c_{\theta_2} & 0 & s_{\theta_2} & c_{\theta_2}+d_3-1 \\ -s_{\theta_2} & 0 & c_{\theta_2} & s_{\theta_2}+d_4-2 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

where θ_2, d_3 and d_4 are the DOF variables of the against mating condition between $f2$ and $f4$.

4.2.4.1. Range Restrictions on DOF Variables

The objective of this section is to combine the range constraints of individual translational DOF variables from the two against mating conditions (d_1,d_2,d_3 and d_4). The combination requires comparing them in the common reference frame. Furthermore, the range constraints of individual DOF variables are derived from the extremes along the x and y axes of the face frame, approximating the

bounds of the face.

Each face is associated with minimum and maximum values along the x and y axes of the frame, affixed to the face feature. For $f1$, δx_1 and δy_1 are the bounding parameters along x and y axes with respect to its own frame respectively as:

$$0 \le \delta x_1 \le 1$$

$$0 \le \delta y_1 \le 0.5$$

Likewise, the bounds of $f2$, $f3$, and $f4$ are derived from the geometric models of the Bench and the Beam where δx_2 and δy_2 for $f2$, δx_3 and δy_3 for $f3$, and δx_4 and δy_4 for $f4$.

$$0 \le \delta x_2 \le 1 \qquad -0.5 \le \delta y_2 \le 0$$

$$0 \le \delta x_3 \le 1 \qquad -0.5 \le \delta y_3 \le 0$$

$$0 \le \delta x_4 \le 1 \qquad 0 \le \delta y_4 \le 1$$

Note, a point is translated at different coordinate systems by the following relationship:

$$\vec{p}_a = H_a^b \vec{p}_b$$

where $\vec{p}_a$ and $\vec{p}_b$ are describing a point in frame a and b respectively.

In order to compare the bounds of $f3$ (δx_3 and δy_3) with those of $f1$ (δx_4 and δy_4), the bounds of $f1$ are transformed to the face frame affixed to $f3$ as the common reference frame, RelFrame1 is a transformation matrix for the frame of $f1$ with respect to $f3_1$ in Fig. 4.3 (a); it is derived by traversing the following nodes in the position map of Fig. 4.4: $f3_1$, $b2_0$, *world*, $b1_0$, and $f1_0$ and collecting the frames in the path.

$$RelFrame\,1 = AgainstFrame\,3^{-1} \times BaseFrame\,2^{-1} \times BaseFrame\,1 \times Frame\,1 = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 3 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

RelFrame1 is the frame of $f1_0$ with respect to that of $f3_1$ and can be visualized in Fig. 4.5. x3, y3, and z3 are axes of $f3_1$ and x1, y1, and z1 are axes of $f1_0$. Using $RelFrame\,1$, δx_1 and δy_1 of $f1$ are

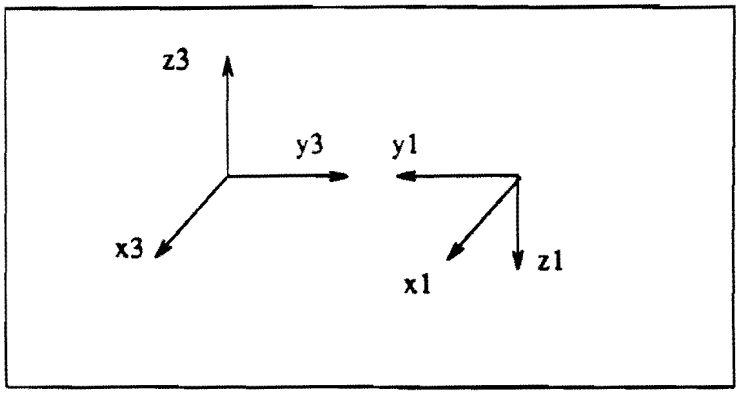


Figure 4.5 Face Frame of $f\,1_0$ with respect to that of $f\,3_1$

translated into $\delta x_1'$ and $\delta y_1'$, the bounds respect to $f\,3_1$. Note, the minimum and maximum points of the bounds are:

$$Minimum = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad Maximum = \begin{bmatrix} 1 \\ 0.5 \\ 0 \\ 1 \end{bmatrix}$$

Translate the maximum bounds with respect to $f\,3_1$ for $\delta x_1'$ and $\delta y_1'$ with respect to $f\,3_1$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 3 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0.5 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2.5 \\ 0 \\ 1 \end{bmatrix}$$

Therefore,

$$0 \le \delta x_1' \le 1$$

$$2.5 \le \delta y_1' \le 3$$

Then, the bounds of the displacements (d_1 and d_2) are derived from the bounds of $f\,3$ and $f\,1$ in the common reference frame of $f\,3_1$ as follows:

$$-1 \leq d_1 = \delta x_1' - \delta x_3 \leq 1$$

$$2.5 \leq d_2 = \delta y_1' - \delta y_3 \leq 3.5$$

Using the SUP-INF method [Bledsoe 1974], the translational displacements of DofFrame3 derived using the above inequalities as the context.

$$-1 \leq p_x(DofFrame3) = d_1 \leq 1$$

$$0.5 \leq p_z(DofFrame3) = d_2 - 2 \leq 1.5$$

Similarly from the against mating condition between $f2$ and $f4$, the translation components are computed using the range context of c_{θ_1} and s_{θ_1} as between -1 and 1:

$$-1 \leq p_x(DofFrame4) \leq 3$$

$$-3.5 \leq p_y(DofFrame4) \leq 0$$

4.2.4.2. Combining the Effects from Multiple Mating Conditions

In general, there is more than one mating condition for a kinematic liaison between two components and it is necessary to combine the kinematic constraints from each mating condition. The method is explained by an example of Fig. 4.3 (a). There, two against mating conditions are specified between the Bench and the Beam. In the previous section, we already analyzed the kinematic constraints from each of the two against mating conditions and the ranges of the displacements were derived.

Note, *DofFrame*3 is a new base frame of the Beam with respect to its initial base frame which mates $f1$ and $f3$. *DofFrame*4 is a new base frame of the Beam with respect to its initial base frame which mates $f2$ and $f4$. Then, the new matrix combining the local effects (*DofFrame*3 and *DofFrame*4) is subsumed by both *DofFrame*3 and *DofFrame*4. That is, each entry of the new entry

should be a refinement of the corresponding entries from *DofFrame* 3 and *DofFrame* 4: an equality is asserted for each corresponding entry and the 12 simultaneous equations are shown in Fig. 4.6.

Before solving the simultaneous set of equations by a numerical method, SRE reduces the form of the equations to simplify the equation solving task by:

constant dominance rule

if one of the corresponding entries is a constant, the constant *dominates* over any expressions except other constants,

constant propagation rule

when a constant corresponds to a single term, all the occurrences of the term are replaced by the constant.

The above rules are used to reduce the form of the equations, simplifying the numerical computation task. The **constant dominance rule** and *constant propagation rule* work in conjunction. In Fig. 4.6, a_y *states that* c_{θ_1} *is one. One dominates* over c_{θ_3}. Using the constant propagation rule for the equality of n_x, c_{θ_1} is one; $d_1 = d_3$ using the rule for p_x.

$n_x: c_{\theta_1} = c_{\theta_3}$	$n_y: 0 = -s_{\theta_3}$	$n_z: s_{\theta_1} = 0$
$o_x: s_{\theta_1} = 0$	$o_y: 0 = 0$	$o_z: -c_{\theta_1} = -1$
$a_x: 0 = s_{\theta_3}$	$a_y: 1 = c_{\theta_3}$	$a_z: 0 = 0$
$p_x: d_1 = c_{\theta_1}+d_3-1$	$p_y: -1.5 = s_{\theta_3}+d_4-2$	$p_z: d_2-2 = 1$

Figure 4.6. Set of 12 Equality Equations

This method layers the problem-solving according to their difficulty. In general, it is necessary to solve non-linear simultaneous set of equations with a numerical method such as the least-square method: a computationally expensive procedure. Here, a more powerful, therefore expensive, numerical method is deployed only when simpler method fails. As a result, reasoning over the form of equations layers the deployment of the computational procedures according to their difficulty.

Furthermore, an inconsistency is detected when two corresponding entries are reduced to different constants. Such inconsistencies are manifested when the component geometries are inconsistent with the assembly geometry designed by the topological specifications: mating conditions. This is a useful feedback for an assembly designer who may modify the component geometry or the assembly specification.

Here, the equations of Fig. 4.6 are reduced to only a single variable, d_1 and the transformation matrix, subsumed by both DofFrame3 and DofFrame4, is:

$$RotTransform(Bench, Beam) = \begin{bmatrix} 1 & 0 & 0 & d_1 \\ 0 & 0 & 1 & -1.5 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note, there is only one translation DOF variable along x axis of the initial base frame, BaseFrame2, d_1: a *prismatic joint*. Using the ranges of $p_x(DofFrame\,3)$ and $p_x(DofFrame\,4)$:

$$-1 \le p_x(DofFrame\,3) = d_1 \le 1$$

$$-1 \le p_x(DofFrame\,4) \le 3$$

The range of d_1 is an intersection of the above two inequalities:

$$-1 \le d_1 \le 1$$

CHAPTER 5.

ASSEMBLY PLANNING FROM THE FIRST PRINCIPLES

When an assembly is being designed, it is necessary to determine whether the design can be realized. This decision can be broken into checking for the *manufacturability* and *assemblability* of the design. Manufacturability can be determined by planning for manufacturing individual components, called a *process planning*; assemblability by planning for assembling the components, called an *assembly planning*. The latter denotes a process of creating a plan structure that can be used to bring the individual components together to satisfy their topological and kinematic relationships, specified by the assembly designer, in the target assembly diagram.

The assembly planning activity requires predicting the consequences of planning each assembly step. This prediction requires the knowledge of an *assembly task theory*. Using the first principles of the physical environment as the task theory, an assembly planner can simulate individual assembly steps and thereby, can predict their consequences. Newtonian mechanics is the first principle of the physical environment. However, using the Newtonian mechanics as an assembly task theory is intractable. In fact, this chapter shows that reasoning even from the restricted theory of the Newtonian mechanics, the collision physics, is intractable with increasing number of components.

This chapter describes the assembly planning from the first principle of the simulated environment, a collision physics, by projecting the effects of planning a mating operation using the collision physics. To overcome the complexity barrier of this approach, the next chapter describes another approach to assembly planning from the previous assembly experiences called *empirical assembly planning*.

5.1. Assembly Planning from Collision Physics

The planner should impose the precedences that would avoid an assembly sequence whose initial subsequence constructs a subassembly that prevents a collision-free path for an assembly step of the latter subsequence. The assembly planning is based on the spatial interferences between components or subassemblies during their mating operation.

The precedence constraints are detected by checking spatial interferences between a pair of subassemblies or components. For example, consider the Bell-Head assembly in Fig. 5.1. Fig. 5.2 shows that the PIN and the RING are already assembled and the HEAD is being assembled by traversing the environment: the striped regions mark the swept volumes of the HEAD. However, the space, occupied by the PIN, intersects with the swept volume of the HEAD in Fig. 5.2 (a) and with the swept volume of the RING in Fig 5.2 (b). Therefore, the HEAD should be assembled before the PIN or the RING.

With this precedence constraint, four assembly sequences are possible: <RING, HEAD, PIN>,

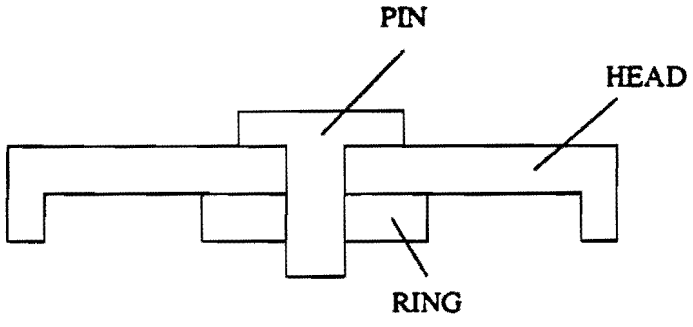
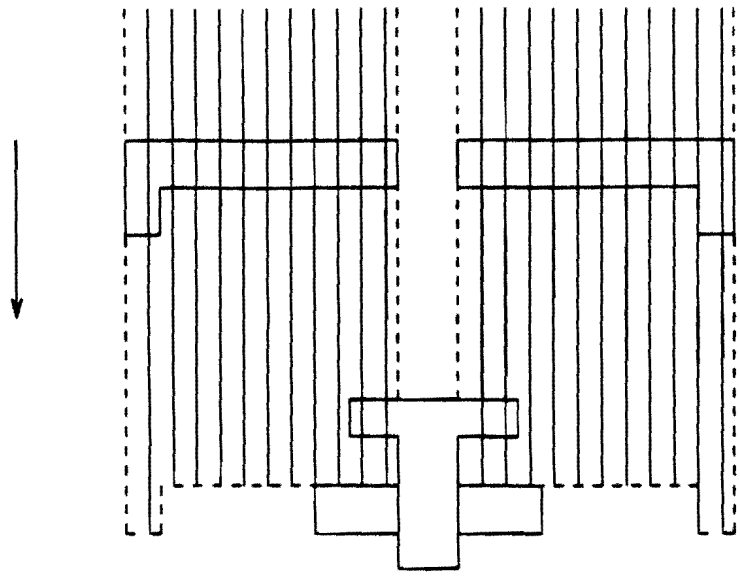
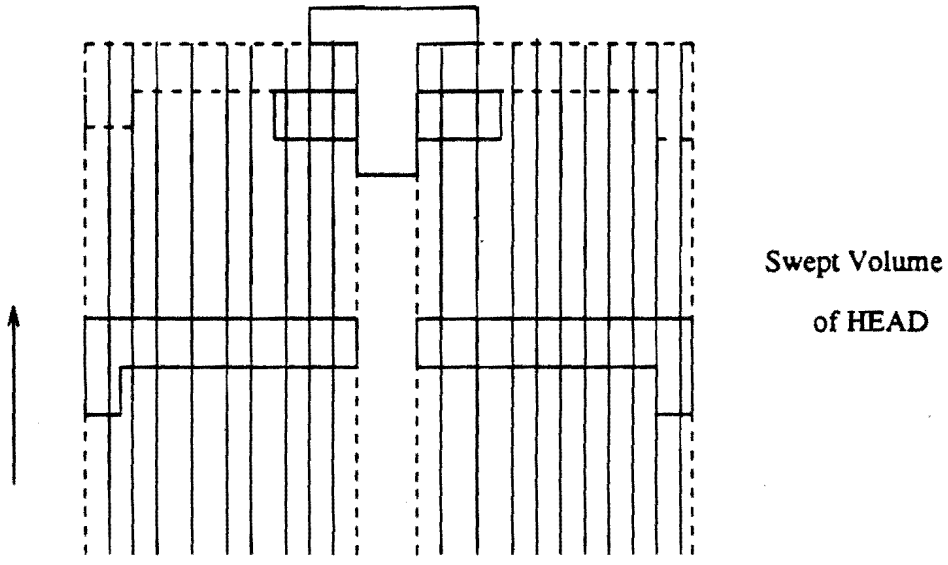


Figure 5.1. Bell-Head Assembly



(a) Interference between HEAD and PIN



(b) Interference between HEAD and RING

Figure 5.2. Assembling HEAD when PIN and RING are in place

<PIN, HEAD, RING>, <HEAD, PIN, RING>, or <HEAD, RING, PIN>[1]. To determine the precedence relations between the PIN and the RING, Fig. 5.3 (a) shows the first sequence and last sequence when the HEAD and the RING are in place and the PIN is being assembled. Because the swept volume of PIN does not intersect, the PIN can be assembled. Likewise, Fig. 5.3 (b) shows the second and third sequence when the PIN and the HEAD are in place, the RING can be assembled. Therefore, both sequences are admissible and no precedences are necessary between assembling the PIN and the RING. Therefore, the only precedence constraint is that the HEAD should precede the PIN or the RING and all the four sequences are admissible.

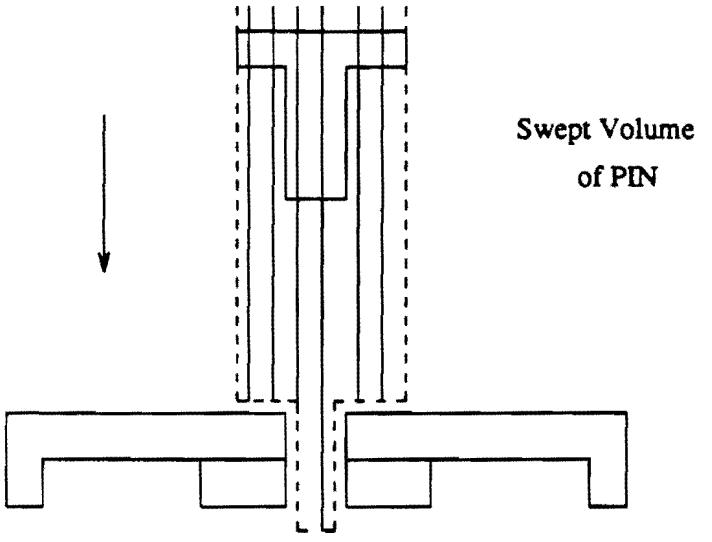
5.1.1. Assemblability from Accessibility

The above interference checking using a swept volume (called spatial envisionment) determines the accessibility of a moving component to its goal position in the presence of other components already assembled. An assembly step is feasible if the final position is accessible. The accessibility is determined by envisioning a collision-free path for the mating operation. When no collision-free path can be found, the planner discovers precedence constraints like in the previous section. Currently, a failure to find the collision-free path is reported by the user as a precedence violation during the envisionment.

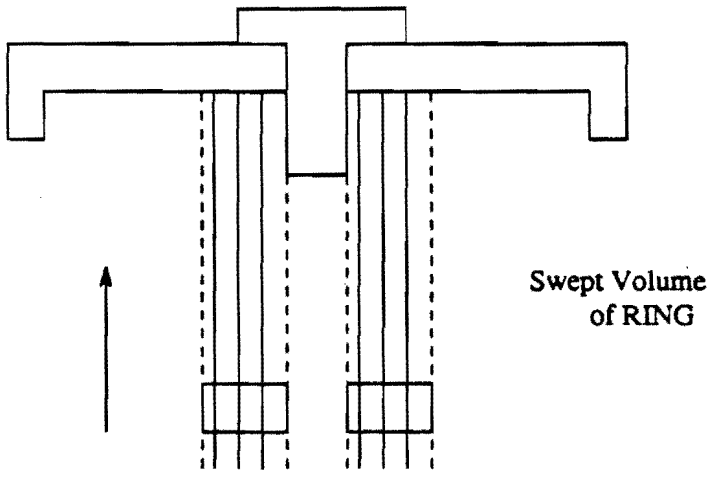
5.1.1.1. Assemblability vs FIND-PATH Algorithm

Computing the collision-free path is related to the FIND-PATH problem [Brooks & Lozano-Perez 1983]. The problem is to find a continuous path for an object with an initial location and orientation to reach the goal location and orientation in the environment, cluttered with the obstacles. Furthermore, the traversal should follow a path without any collisions with the obstacles, a collision-

[1] <HEAD, RING, PIN> represents a sequence of components, being assembled.



(a)



(b)

Figure 5.3. Interference Checkings between PIN and RING

free path.

For the assembly planning problem, a component of a target assembly is moved to reach its assembled position as a goal position by traversing the environment, occupied by the assembly components that are already in their assembled positions. Other components in their initial positions are located where they do not interfere with any mating operations so that any interferences that are detected during the assembly operations are attributed to the components already in their goal (assembled) positions. Then, the objective of an assembly planning is to devise a sequence of changes to the environment so that a collision-free path can be found for each assembly operations by a FIND-PATH algorithm.

Because a comprehensive FIND-PATH algorithm shall extend the notion of accessibility, the notion of the *assemblibility* by the assembly planner depends on the FIND-PATH algorithm used by the planner. When no assembly sequence for a target assembly can be constructed with limited FIND-PATH algorithm for the interference checking, it may be possible to construct an assembly sequence for the target assembly with more powerful FIND-PATH algorithm. As a result, unless the FIND-PATH algorithm is complete, a failure to find an assembly sequence for a target assembly does not imply the target assembly is not intrinsically assemblable.

Although there exists an algorithm, solving the FIND-PATH problem completely in 3D with all six degrees of freedom (three translational and three rotational), the algorithm is not polynomially bound [Donald 1984]. Hence, currently, any approach to assembly planning with a comprehensive FIND-PATH algorithm is computationally intractable. To compound the difficulties, the number of spatial interference checkings grows exponentially with the number of components, as shown in the next section.

5.1.2. Analysis of The Number of Interference Checkings

This section analyzes the maximum number of interference checkings with N components. Consider the worst case by assuming any subset of the components forms a subassembly. When i components are already assembled, a spatial interference should be checked with any subset of the rest of $N-i$ components except the empty set: $2^{N-i}-1$ spatial interference checkings as shown in Fig.5.4.

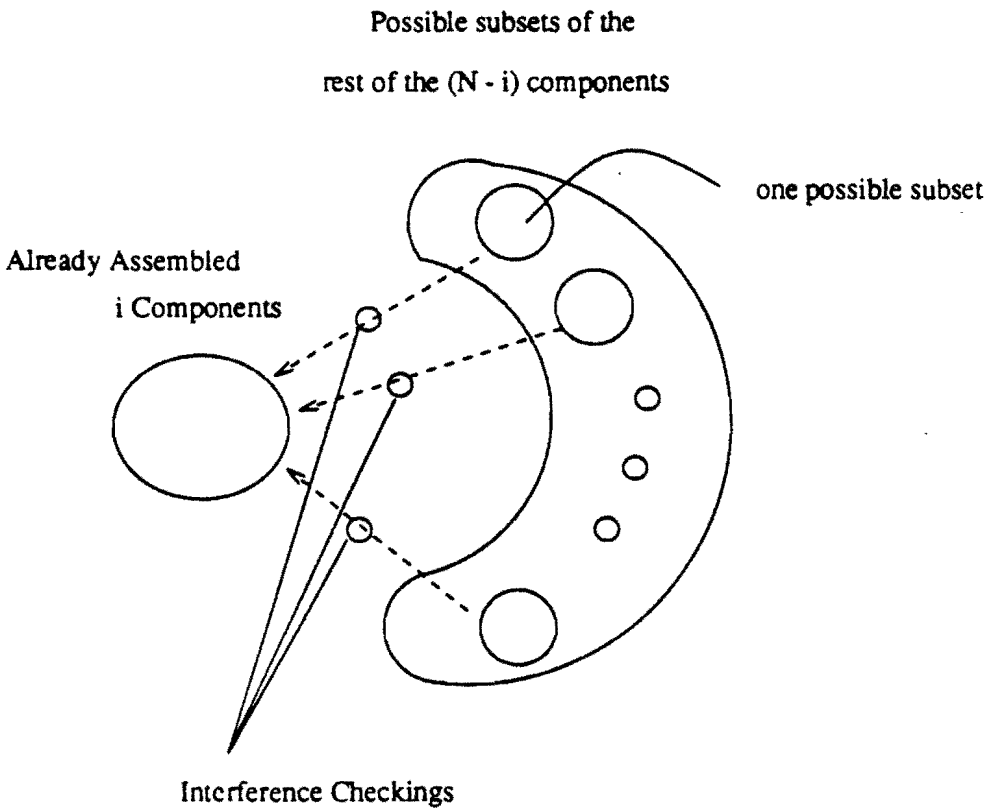


Figure 5.4. Number of Spatial Interference Checkings with i Components

Because there are $\binom{N}{i}$ possible assemblies with i components that are already assembled, there are $\binom{N}{i}(2^{N-i}-1)$ spatial interference checkings. Note, this complexity analysis assumes that assembling a subassembly, A, to another subassembly, B, requires a separate test from B to A. Finally, the total number of spatial interference checkings is:

$$\sum_{i=1}^{N-1}\binom{N}{i}(2^{N-i}-1)$$

Solve the above formula using the binomial theorem as follows:

$$\begin{aligned} 3^N &= (1+2)^N \\ &= \binom{N}{0}1^0 2^N + \binom{N}{1}1^1 2^{N-1} + \ \ldots\ + \binom{N}{N}1^N 2^0 \\ &= \sum_{i=0}^{N}\binom{N}{i}2^{N-i} \end{aligned}$$

Similarly,

$$\sum_{i=0}^{N}\binom{N}{i} = 2^N$$

Using the above equations, solve:

$$\begin{aligned} &\sum_{i=1}^{N-1}\binom{N}{i}(2^{N-i}-1) \\ &= \sum_{i=1}^{N-1}\binom{N}{i}2^{N-i} - \sum_{i=1}^{N-1}\binom{N}{i} \\ &= (3^N-2^N-1)-(2^N-2) \\ &= 3^N-2^{N+1}+1 \end{aligned}$$

Therefore, the total number of spatial interference checkings grows exponentially with respect to N, the number of components.

5.1.3. Assembly Hierarchy as an Abstraction Hierarchy

In order to limit the number of the spatial interferences, the assembly problem should be decomposed into pieces. The pieces are more or less disjoint and they together cover the entire problem. For each subproblem, it is further decomposed until the problem can be easily solved. This decomposition process constructs a hierarchy of assembly subproblems

Considering the target assembly as a goal expression, the individual goals are hierarchically organized and each level forms an an abstraction level so that the planner considers only the individual goals that are important at a given level of abstraction. This process forms a tree-like decomposition of the goal expression, a goal hierarchy.

The goal hierarchy is formed by decomposing the target assembly into subassemblies and each subassembly is further decomposed into smaller subassemblies forming an assembly hierarchy. Therefore, a goal hierarchy is synonymous with an assembly hierarchy. An assembly hierarchy represents the abstraction hierarchy of the assembly problem correctly when the sibling subassemblies are independent (disjoint). With a correct assembly hierarchy, the planner checks only the siblings of a node in the assembly hierarchy because no interference checkings are necessary for the children of the sibling subassemblies if the hierarchy is correct. With a correct hierarchy, the number of spatial interference checkings is reduced dramatically.

Suppose N (2^M-1 for some integer M) components are hierarchically organized as the nodes of a binary tree, as shown in Fig. 5.5. For each internal node (a node with children), its two children are ordered by a spatial interference checking. Then, the total number of spatial interference checkings is the number of internal nodes that is linear with respect to the number of components, N while it is exponential in general as shown in the previous section ($3^N-2^{N+1}+1$):

$$2^{M-1}-1 = \frac{N-1}{2}$$

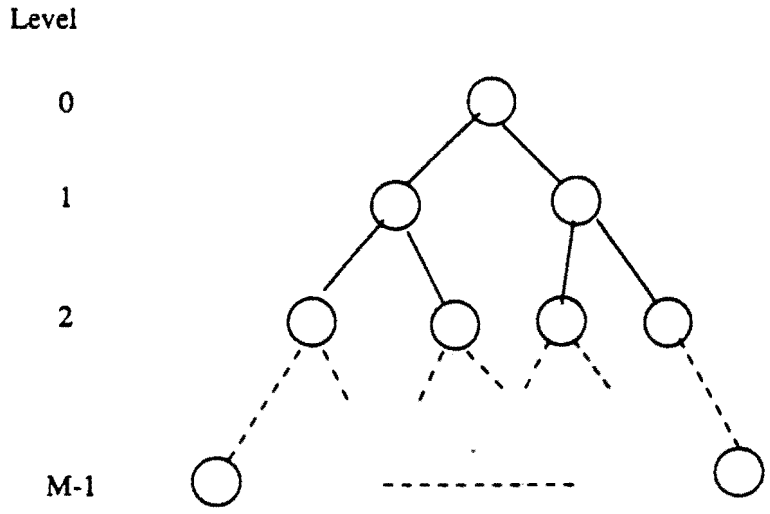


Figure 5.5. Binary Tree

5.1.3.1. Abstraction Principles of the Assembly Hierarchy

With an assembly hierarchy, the final assembly sequence should be derived from the structure of the assembly hierarchy, reflecting the hierarchical structure, as well as from the collection of the assembly subsequences determined for each subassembly. The hierarchical structuring is reflected by additional precedences, derived from a dictionary definition.

A subassembly should follow two principles: *Subassembly Precedence Constraint* (SPC) and *Subassembly Independence Constraint* (SIC). SPC states that the assembly steps of the assembly sequence for a subassembly ("assembly steps in a subassembly" for short) should precede those in the parent assembly. SIC states the assembly steps between sibling subassemblies in the assembly hierarchy are carried out independently. If it is necessary to interleave assembly steps between sibling subassemblies, then they are not independent.

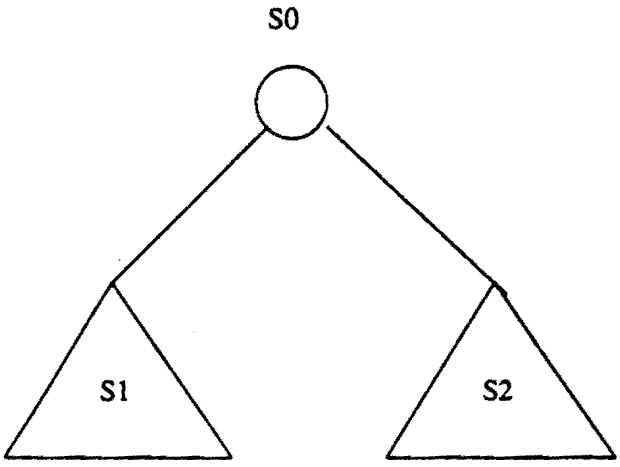


Figure 5.6. Schematic Diagram of an Assembly Hierarchy

Schematically, suppose S1 and S2 are subassemblies of S0 as shown in Fig. 5.6. Here, we incorporate more specialized version of SIC: if NOMAD starts constructing a subassembly, it completes the construction of the subassembly and all of its subassemblies. That is, if S1 is being constructed and S2 is not completed yet, then none of the assembly steps of S2 can be carried out during the construction of S1. Therefore, the assembly steps of S2 should follow those of S1. This interpretation of SIC can be logically expressed as:

$$\exists\ assembly\text{-}step1 \in S1,\ assembly\text{-}step2 \in S2 \mid assembly\text{-}step1 < assembly\text{-}step2 \supset \forall\ assembly\text{-}step3 \in S1,\ assembly\text{-}step4 \in S2 \mid assembly\text{-}step3 < assembly\text{-}step4$$

Likewise, SPC can be logically expressed as:

$$\forall\ assembly\text{-}step \in S0,\ subassembly\text{-}step \in S1 \cup S2 \mid subassembly\text{-}step < assembly\text{-}step$$

where "<" represents a precedence relation between assembly steps: step1 < step2 means step1 should precede step2. The assembly planner uses SPC and SIC to introduce additional precedences

reflecting the structure of the assembly hierarchy as additional precedences in the final assembly sequence.

5.1.4. Related Work: Planning with a Goal Hierarchy

The assembly hierarchy is a goal hierarchy for the planner to decompose the planning tasks hierarchically. Many previous planning systems have used the term "goal hierarchy". This section disambiguates them in comparison to the use of the assembly hierarchy as an abstraction hierarchy.

5.1.4.1. Subgoal Hierarchy

Most STRIPS-type planners construct a goal hierarchy by a goal regression mechanism. A regression of a goal over an operator spawns subgoals that when satisfied, would guarantee that applying the operator will achieve the goal. A goal regression stops when all the subgoals are achievable in the initial situation. This goal hierarchy can better be names as a "subgoal hierarchy" because the subgoal hierarchy is not an abstraction hierarchy of the original goals.

5.1.4.2. Linear Approximation

The linear approximation of the conjunctive goals [Sussman 1973] is to make believe that each conjuncts are independent and they will not interact, hence individual goals are separately planned for in sequence and their individual plans are assumed to be additive by concatenation. Here, the goal hierarchy for the conjunctive goals is a linear sequence. In a tree structure, the first goal of the sequence is the only leaf, the last goal is the root with only one path from the root to the leaf.

The linear approximation can only create a fixed sequence of goals for the goal hierarchy. Here, the goal hierarchy is a tree structure, supporting partial ordering of goals and can be built dynami-

cally. Note, the non-interaction assumption between goals in the linear approximation is the SIC for the assembly hierarchy.

5.1.4.3. Criticality Value Assignment

ABSTRIPS [Sacerdoti 1974] assigns a criticality value to each predicates of a problem domain a priori. When planning for a problem, multiple goals were planned according to their predicate criticality value. This planning paradigm is called a *hierarchical planning* because the problems, denoted by the highest criticality value predicates, are planned for before planning for predicates with lower criticality value. However, the criticality values were fixed across different problem-solving episodes.

Therefore, the individual goals can be partially ordered, extendeding the linear ordering of goals by the linear approximation above. However, ABSTRIPS can only construct a single assembly hierarchy, so to speak, after giving the same problem many times because its criticality value does not change with the amount of the experience. Furthermore, the criticality is assigned either by a domain independent (weak) fixed strategy[2] or by the user. Furthermore, if the problem contained many predicates with same criticality values, ABSTRIPS resorted back to the linear approximation.

In the assembly task domain, the goal description is the structural and kinematic description of the target assembly diagram: the mating conditions. Because there are only limited number of different types of mating conditions, the criticality values among the mating conditions would not distinguish among the same type of mating conditions. Therefore, the criticality assignment would not introduce much hierarchical structuring over the problem space. Furthermore, it is incorrect to assign a fixed criticality value to the whole type of mating condition because it is often that the aligned condition should precede against condition and vice versa depending on the assembly situation.

[2]The predicates that can not be satisfied by any available operators are assigned the highest criticality values.

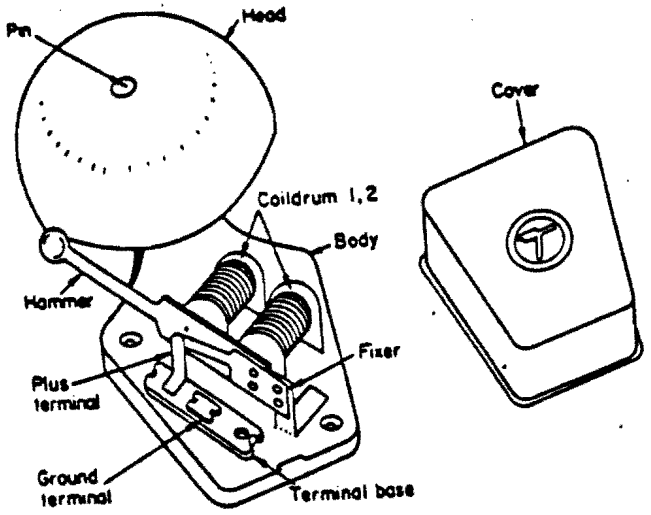
NOMAD constructs the assembly hierarchy by two different methods. The next section illustrates a hierarchy formation using a domain dependent heuristic based on the topological structure of the target assembly diagram. The next chapter will describe a hierarchy formation based on the the previous assembly experiences. As a result, different hierarchies are formed with the different experiences. In terms of the criticality assignments, the dynamic hierarchy formation above is equivalent to the criticality values and their effectiveness from one planning episode are transferred to other similar assembly tasks in the future.

5.1.5. Case Study: Bell-Assembly

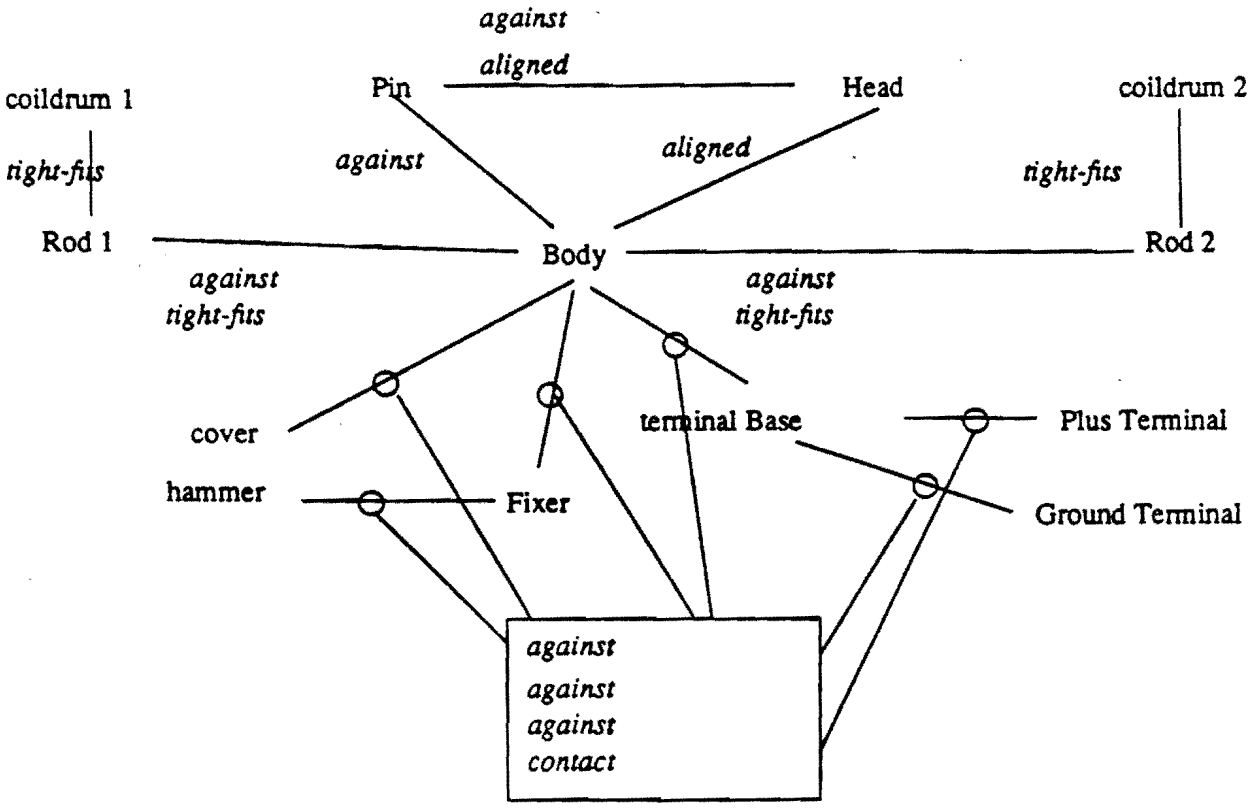
The collision-physics based assembly planning approach was applied to a moderate size target assembly with 13 components, the Bell-assembly shown in Fig 5.7 (a). Its target assembly diagram is described in Fig. 5.7 (b). The diagram shows additional mating conditions that were not explained in Chapter 3: *tight-fits* and *contact*. *Tight-fits* condition specifies an *aligned* condition with no rotational degrees of freedom; *contact* condition is an ad hoc mating condition to prevent any relative degrees of freedom between mating features. These mating conditions require kinetic semantics but the kinetics is beyond the scope of the simulated environment. However, they are introduced to cope with this example. Therefore, their applicability is limited to this example.

5.1.5.1. Assembly Hierarchy Formation

The assembly hierarchy formation is based on the target assembly diagram. Identify the component, called a *base component*, which is connected to the gratest number of components by the kinematic liaisons of the non-subassembly type. A kinematic liaison is classified as a subassembly type when the mating components has any relative degrees of freedom. Gather all the components connected to base component by the kinematic liaison of the non-subassembly type. Call the gath-



(a)



(b) Target Assembly Diagram
Figure 5.7 Bell-Assembly

ered components as *satellite components* of the base component, *satellites* for short. The collection of the base and satellite components is called a *grouping*.

The grouping formation by gathering satellites is *recursively* applied with each satellite of the grouping as a new base component. Therefore, the groupings are hierarchically organized. Same procedure is applied for the components not covered by any of the groupings.

Using the target assembly diagram of Fig. 5.7 (b), Body is chosen because it is connected by greatest number of kinematic liaisons, five. The kinematic liaisons between Body and Head and Pin are subassembly type. The satellites of Body are Cover, Rod 1, Rod 2, Fixer, and Terminal Base. A grouping is formed for each satellite as a base. The Terminal Base as a base component collects Ground Terminal and Plus terminal as its satellites. Likewise for Fixer, Cover, Rod 1, and Rod 2. When this process has not yet covered Head and Pin. Choose any because neither of them are connected by a kinematic liaison of the subassembly type. Head is a base component with no satellites; likewise for Pin. This process results in the component hierarchy in Fig. 5.8 as its assembly hierarchy.

5.1.5.2. Assembly Sequence Generation

All the precedence relations of the Bell assembly is generated from the Bell assembly hierarchy in Fig. 5.8. From the SPC, the assembly hierarchy provides the following precedence constraints:

[Terminal Base, Ground Terminal] < [Body, Terminal Base]
[Terminal base, Plus Terminal] < [Body, Terminal Base]
[Fixer, Hammer] < [Body, Fixer]
[Rod 1, Coildrum 1] < [Body, Rod 1]
[Rod 2, Coildrum 2] < [Body, Rod 2]

where [Body, Terminal Base] means the assembly step to establish the mating conditions of the kinematic liaison between Body and Terminal Base.

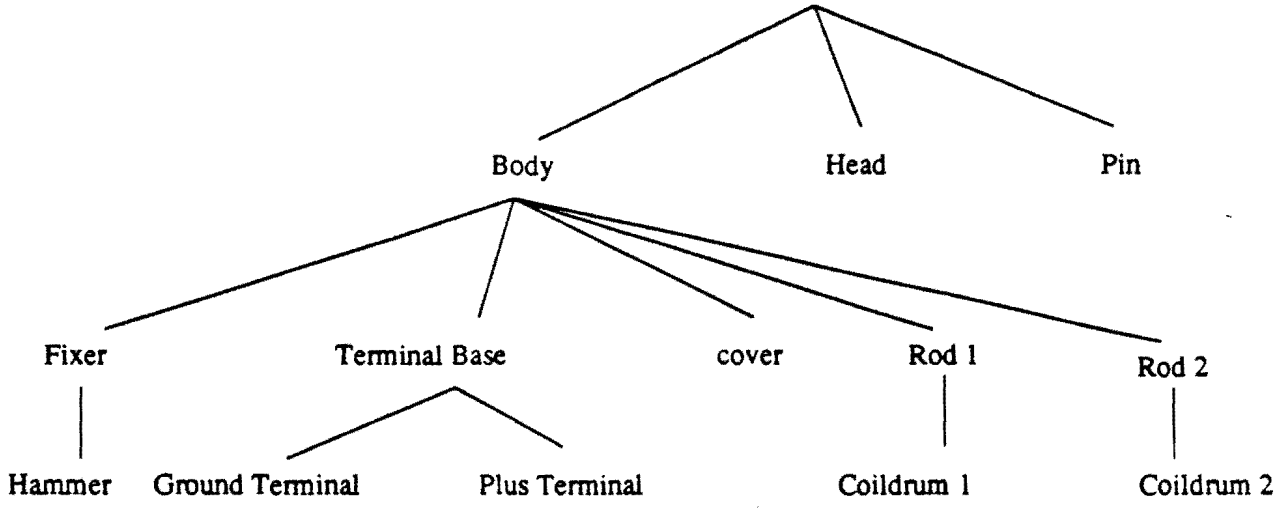


Figure 5.8 Component Hierarchy of Bell Assembly

Then, precedence relations between assembly steps for satellites are found by their pairwise spatial interference checkings in post order. No interferences are detected between the assembly steps for Plus and Ground Terminals. For the Body grouping, the spatial interferences are checked between the assembly steps for the satellites. Compare the assembly steps for Terminal Base and for Cover. Assembly steps for the grouping for Terminal Base (Terminal Base, Plus Terminal, and Ground Terminal) should precede that of Cover because when cover is assembled to the Body, the grouping can not be assembled to the Body.

[Body, Terminal Base] < [Body, Cover]

Similarly, the rest of the precedence relations are discovered:

[Body, Rod 1] < [Body, Fixer]
[Body, Rod 1] < [Body, Terminal Base]
[Body, Rod 1] < [Body, Cover]
[Body, Rod 2] < [Body, Fixer]
[Body, Rod 2] < [Body, Terminal Base]
[Body, Rod 2] < [Body, Cover]
[Body, Fixer] < [Body, Terminal Base]

For comparing Head, Body, and Pin, consider Body as the base because it is the largest object and therefore, most cumbersome to move. Then, similar analysis as the Bell-Head Assembly of Fig. 5.1 except that RING is replaced by Body.

[Body, Head] < [Body, Pin]

Using the transitivity property of "<" relation, the final assembly sequence can be generated from the precedences found for the Bell assembly.

CHAPTER 6.

EMPIRICAL ASSEMBLY PLANNING AND EXECUTION

The complexity reduction via the assembly hierarchy in the previous chapter is predicated on the "correctness" of the assembly hierarchy: all its subassemblies satisfy the SIC and SPC criteria for generating the final assembly sequence. The heuristic formation of the assembly hierarchy is "heuristic: it may produce incorrect assembly hierarchies.

Unfortunately, there is no general theory that would construct an assembly hierarchy correctly without checking spatial interferences between subassemblies of the hierarchy. There are two approaches. One is to ask the user to specify the subassemblies during the assembly design phase as included parts like specifying the components. However, this would simply push many of the planning tasks to the user. Instead, this chapter investigates another approach to generate the hierarchy by neither being told nor using a fixed set of heuristic hierarchy formation strategies. The difficulty with the second approach is:

(1) in order to reduce the number of spatial interferences, an assembly hierarchy should be constructed and

(2) in order to ensure the correctness of the assembly hierarchy, the spatial interferences should checked for the subassemblies.

This cyclical dilemma is overcome by the *empirical assembly planner* that is not concerned with constructing the "correct" assembly hierarchy and sequence everytime. Rather, the assembly planner of this chapter is concerned with learning to construct correct assembly hierarchies and sequences eventually after trial-and-error in many planning episodes.

Each assembly construction is assimilated as assembly scenarios in memory. Using the compiled knowledge of the assembly scenarios, the assembly planner avoids duplicating assembly hierarchy formations that failed; thereby, an assembly hierarchy formation in the future will be correct for more cases than before. This chapter presents the planning and execution processes by the empirical assembly planner.

6.1. Empirical Planning Method

This section presents an *empirical planning method* that is used as a basis for developing the empirical assembly planning. Empirical planning (EP) involves both planning and learning. As a planner, EP generates a plan for the given task situation from the store of schemas in memory. The schemas are of two types: one is a generalized description of a task situation and the other is a generalized description of a plan segment that can be used to achieve some task situation. These two types of schemas are interrelated forming pairs: each pair consists of a schema for the plan segments (plan schema) and another for the task situations the plan segments achieve (task schema).

Given a task situation and store of schemas in memory, the planning process proceeds in three steps:

(1) using the task schemas in memory, form subtask situations of the given task situation,

(2) post the plan segments of the subtask situations as candidate subplans,

(3) combine the subplans into a coherent plan for the given task situation.

A task schema in memory is instantiated with respect to the task situation. Each instance represents a subtask situation for which plan segments are generated by instantiating the plan schemas associated with the task schema. These subtask situations form decompositions of the given task situation and their plan segments represent candidate subplans. These subplans are posted to

determine those that are mutually inconsistent because they can not be combined into a coherent plan. The subtask situations are partitioned into maximally consistent sets. The subplans of the subtask situations of a consistent set are mutually consistent.

Given the consistent set above, EP combines the subtask situations of the set into a hierarchy. The hierarchy represents a hierarchical decomposition of the task situation or an abstraction hierarchy. Using the hierarchy, the planner generates the final plan by combining the subplans according to the abstraction hierarchy.

The planner's predicted success of the final plan may fail when the schemas used by the planner are incorrect: EP does not assume the correctness of the schemas in memory. EP monitors the execution of the final plan to empirically validate or to invalidate the schemas used by the final plan. As a learner, EP assimilate the empirical validations and invalidations of the schemas in memory. The learning task is to accumulate empirically correct and complete schemas that can be used for the range of task situations to be encountered.

6.2. Empirical Assembly Planning/Execution

EP method is applied for the assembly task situations called *emprical assembly planning*. The empirical assembly planner creates an assembly sequence from the assembly scenarios, the empirical observations of what schemas has and has not worked before. An assembly planning generates the final assembly sequence by the memory instantiation of these schemas and the manipulation of the instances. The planner monitors the results of executing the final assembly sequence. The success of executing a plan segment, posted by an assembly planning scenario, of the final assembly sequence empirically validates the scenario while the failure empirically invalidates the scenario. These empirical observations are used by the learning element to modify the schemas. This section discusses the assembly planning and execution phase. The next chapter will describe the learning mechanisms in

detail.

6.2.1. Assembly Planning Episodes and Scenarios

An assembly planning episode is a building block by the assembly planning and an assembly scenario is a canonical chunk of the memory from which assembly planning episodes are created. This section describes the representation of assembly planning episodes and scenarios and the assembly planning episode formation. The assembly scenario formation is described in the next chapter.

6.2.1.1. Assembly Planning Episode

A *grouping* is a task situation describing a target assembly that can be assembled as a unit. The assembly task can be achieved by fixing one component of the grouping, called a *base*, and by mating the rest of the components of the grouping, called *satellites*, to the base in sequence according to an assembly sequence. Each step of the assembly sequence specifies a mating operation between a satellite and the base. An assembly sequence is a potential plan segment of the final assembly sequence.

The relationship between a grouping and an assembly sequence is maintained by a link. A pair of a grouping and an assembly sequence represents an *assembly planning episode*. Note, by executing an assembly sequence, a grouping is formed and the grouping is unique with respect to the kinematic degrees of freedom of the structure: that is, different configurations of the grouping are congruent as long as they satisfy all the kinematic constraints of the grouping. However, there may be more than one assembly sequence that can be used to construct the grouping. Therefore, a grouping may be linked to more than one assembly sequence.

Fig. 6.1 diagrams a grouping with $n+1$ components with an assembly sequence: assemble from Sat-1 to Sat-n in sequence. A circle represents a component. The circles are labeled by mnemonic

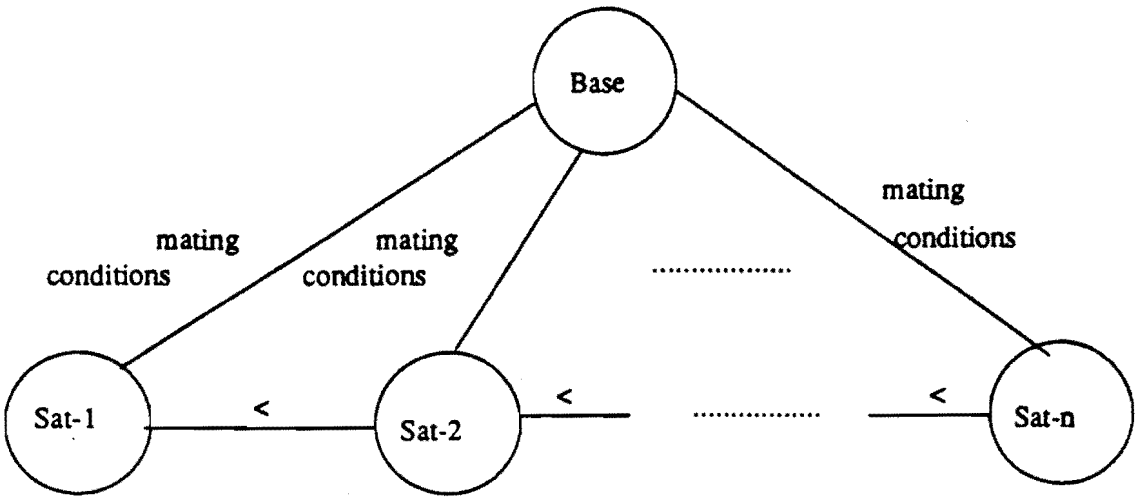


Figure 6.1. Diagram for an Assembly Planning Episode

abbreviations according to the roles of the components in the grouping. The circle with "Base" represents a base component and the circles with "Sat-1", "Sat-2", and "Sat-n" represent satellite components.

The assembly sequence associated with the grouping is shown in the diagram by the edges labeled by "<" between satellite components. This is a shorthand: the precedence relations are between the assembly steps for satellites to be mated with the base component. However, an assembly sequence can be unambiguously represented by ordering just the satellites in the context of a grouping. "Sat-1" is the first satellite to be assembled to the "Base" and "Sat-n" is the n-th (also the last) satellite to be assembled.

The edges, labeled by mating conditions, represent the kinematic liaison between components. The mating conditions and the components represent structural content of an assembly planning episode and the assembly sequence represents procedural content. The diagram integrates the

structural content, the base component, satellite components, and their mating conditions, with the procedural content, a sequence of satellites. The structural content belongs to a grouping and the procedural content belongs to an assembly sequence as a plan segment for the grouping. The mating conditions between satellites in parentheses are included in the assembly sequence because they specify kinematic constraints between satellite components although they represent structural content of an assembly planning episode.

Consider the Bell-Head assembly of Fig. 5.1 as a target assembly. A final assembly sequence is formed by fixing the PIN as a base and assembling the HEAD to the PIN and then, assembling the RING. This assembly planning episode is represented by the diagram in Fig. 6.2.

6.2.1.2. Assembly Scenario

An *assembly scenario* is a generalized description of assembly planning episodes. The structure of a scenario reflects those of the assembly planning episodes. A scenario is represented by a pair of

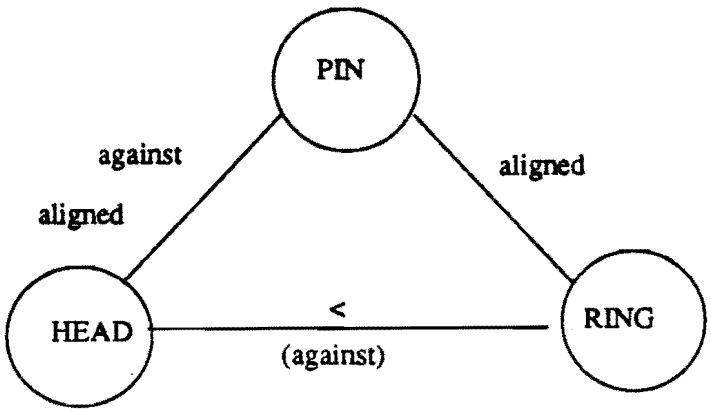


Figure 6.2. Bell-Head Assembly Planning Episode

a grouping schema and a precedence schema. A grouping schema of an assembly scenario is a generalized description of the grouping of the assembly planning episode and a precedence schema is a generalized description of the assembly sequences. A pair of a grouping and a precedence schema represents a unit of an assembly experience, called an assembly scenario.

Grouping and precedence schemas have inner structure like the objects they are describing. A grouping schema describes a collection of groupings. A grouping schema contains a component schema for the base, a base schema, as a generalized description of all the base components of its groupings, the component schemas for the satellites of its groupings (base and satellite schemas, respectively), and schemas for their mating conditions. Inversely, a grouping may be described by more than one grouping schema where the grouping is shared by multiple grouping schemas.

Precedence schemas as its grouping describes a collection of assembly sequences. A precedence schema is associated with a grouping schema by a link to specify the order of mating operations for its satellite schemas like the link between the assembly sequence and a grouping.

An assembly scenario is represented by a diagram resembling the diagrams of their instances as assembly planning episodes above. For example, a diagram for the schemas of the Bell-Head assembly planning episode is shown in Fig. 6.3. $Base is a base component schema, a generalized description of a base component, PIN. $Sat-1 and $Sat-2 are the component schemas for the satellites: HEAD and RING.

Like the assembly planning episode, the precedence relations between assembly steps for the satellite schemas are shown by ordering between satellite schemas in the diagram by "<" label. The grouping schema in Fig 6.3 shows the precedences relations between the assembly steps for $Sat-1 and for $Sat-2 by an ordering between them and the mating condition between them in parentheses.

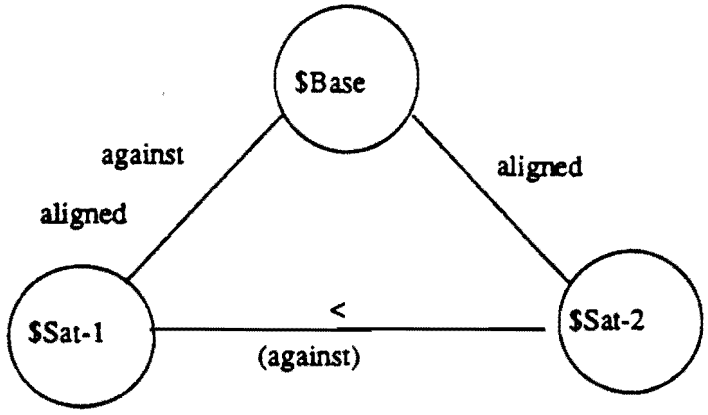


Figure 6.3. Bell-Head Assembly Scenario

6.2.1.3. Assembly Planning Episode Formation

Assembly planning episodes are formed either by instantiating the assembly scenarios in memory or by the user. The user may inform the planner of the groupings and their assembly sequences through question/answering session, the assembly planning episodes. This is necessary when the memory contains incomplete assembly scenarios to cover the new task situation. Otherwise, the schemas are instantiated to generate assembly planning episodes.

Given a target assembly, the planner first forms groupings by instantiating the grouping schemas with the substructures of the target assembly. These groupings decompose the problem of assembling the whole target assembly into subproblems for assembling the groupings. For each grouping, the precedence schemas linked to its grouping schema are instantiated in the context of the bindings for the grouping to form assembly sequences for the grouping. They represent alternative plan segments that can be used to plan for assembling the grouping. Each pair of a grouping and an assembly sequence as an assembly planning episode is a building block for the assembly hierarchy

formation.

For example, the grouping schema of the Bell-Head assembly scenario can be instantiated by the following bindings:

$Base <-> PIN

$Sat-1 <-> HEAD

$Sat-2 <-> RING

The precedence relation of the schema states:

[$Base, $Sat-1] < [$Base, $Sat-2]

Using the above bindings, the following assembly sequence is formed:

[PIN, HEAD] < [PIN, RING]

This Bell-head assembly planning episode is diagramed in Fig. 6.2.

6.2.2. Assembly Hierarchy

The groupings represent separate decompositions of the original problem -- assembling the target assembly. These decompositions should be hierarchically organized to specify the hierarchical decomposition of the original problem, forming an assembly hierarchy. An assembly hierarchy is used by the planner as a goal hierarchy. This section describes the representation and formation of the assembly hierarchies.

The representation of the assembly hierarchy as a goal hierarchy should facilitate the reasoning task by the planner. This criterion determines the representational choices for an assembly hierarchy. This next section describes the pros and cons of two different representational choices for the assembly hierarchy: component-based and liaison-based hierarchy representations.

6.2.2.1. Component-Based Representation

Intuitively, an assembly hierarchy represents a component hierarchy: that is, the assembly hierarchy is composed of components and subassemblies and a subassembly is composed of components and smaller subassemblies. The component hierarchy is natural when planning an assembly operation because each assembly operation is moving a component at a time and an assembly operation can be associated with each component of the hierarchy. For that reason, the case study for the Bell assembly in the previous chapter constructs the assembly hierarchy according to the component-based representation. Fortunately, none of the components are shared across different sibling subassemblies. Unfortunately, such a representation of the assembly hierarchy often results in an entangled hierarchy, not a tree.

6.2.2.1.1. Entangled Hierarchy

A subassembly is designed as a functional unit: it supports some subfunctions of the overall function. Also, it is designed as a unit of assemblage. These dual roles determine subassemblies in the assembly design.

It is often a good design practice for a single component to serve multiple functional roles, the "function sharing" [Ulrich & Seering 1988]. As a result, multiple subassemblies as functional units often share common components because they serve multiple functions from each subassembly. With the common components, a component-based hierarchy often forms an entangled hierarchy. Unfortunately, an entangled hierarchy is ill-suited as a goal hierarchy for the assembly planner.

For example, consider the target assembly in Fig. 6.4. Suppose the components 1, 3, and 4 belong to the subassembly-1; the components 2, 4, and 5 to the subassembly-2. The component-based assembly hierarchy is shown in Fig. 6.5. The component 4 is shared by the subassembly-1 and subassembly-2.

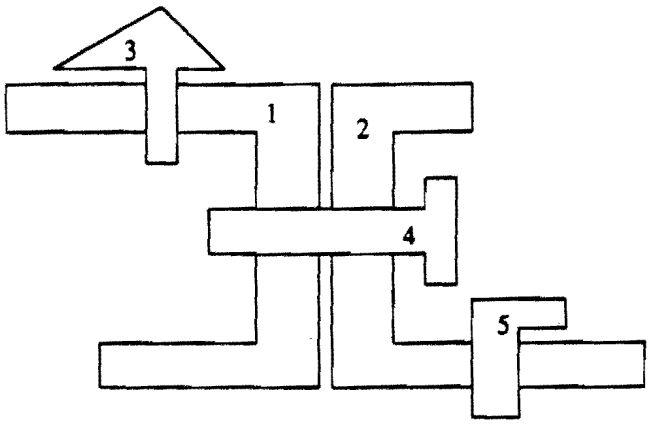


Figure 6.4. Subassemblies

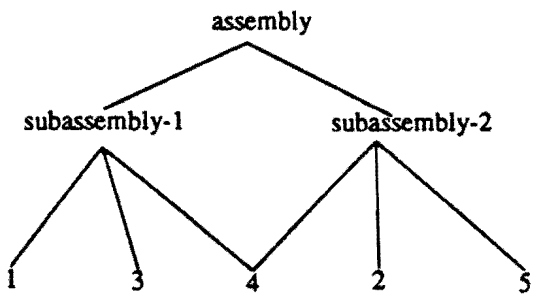


Figure 6.5. Component-based Assembly Hierarchy

The planner considers the assembly hierarchy as a goal hierarchy and assembles subassembly-1 to subassembly-2 as assembly units. Subassembly-1 is assembled as a unit while subassembly-2 is stationary as a unit (all of its components are stationary). This is impossible because the component 4 should be moving and stationary simultaneously during the assembly operation. Hence, the planner

should make these two subassemblies *disjoint* by excluding the component 4 from the subassembly-1 to make it stationary or by excluding from the subassembly-2 to make it assemblable as part of the subassembly-1. Both operations will partially destroy the assembly hierarchy. Therefore, it is cumbersome to treat a component-based assembly hierarchy as a goal hierarchy.

6.2.2.2. Liaison-Based Representation

Instead, the assembly hierarchy is represented in terms of the kinematic liaisons in the target assembly diagram: a subassembly consists of liaisons and smaller subassemblies. Even though two subassemblies may share a common component, the assembly hierarchy maintains a tree form as long as each subassemblies contains distinct kinematic liaisons. The good design practice of "function sharing" does not encourage the sharing of kinematic liaisons for the subassembly design. Furthermore, this investigation suggests subassemblies should be designed so that they do not share kinematic liaisons.

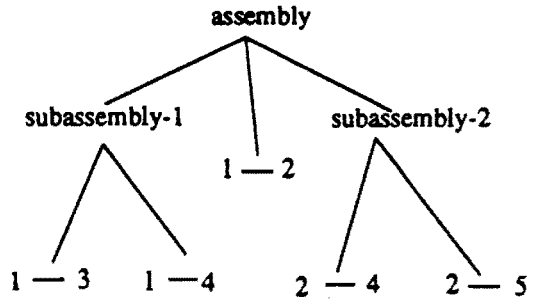


Figure 6.6. Liaison-based Assembly Hierarchy

The liaison-based assembly hierarchy of Fig. 6.5 is shown in Fig. 6.6 for a comparison: a tree structure even though the component 4 is shared between subassembly-1 and subassembly-2. Although components are shared across multiple subassemblies, a tree like structure results as long as kinematic liaisons to the common component are not shared among them. Here, although the component 4 is shared between subassembly-1 and subassembly-2 but their two kinematic liaisons, 1-4 and 2-4, are distinct.

The liaison-based assembly hierarchy representation is based on the virtual-link based representation of an assembly hierarchy that was developed in the context of an assembly design [Lee & Gossard 1985]. A virtual link may link between subassemblies, components and any combinations of them. As a result, the subassemblies in an assembly hierarchy consist purely of virtual links. On the other hand, because a kinematic liaison may link between only components, a subassembly in an assembly hierarchy consists of two distinct types, kinematic liaisons and subassemblies. The schematic diagram of the assembly hierarchy is shown in Fig. 6.7.

Fig. 6.7 shows a general schema for an assembly hierarchy. Each box represents nodes of the hierarchy: internal nodes (nodes with children) and external nodes (nodes without any children). Here, an external node is a kinematic liaison of the target assembly diagram; an internal node represents a subassembly (a root subassembly is called an assembly). This schematic digram represents the internal structure of a subassembly node: two fields for the liaisons and its children subassemblies. These distinctions are implicitly represented by the two types of edges in the assembly hierarchy diagram. One type of an edge connects between subassemblies and another between a subassembly and a kinematic liaison.

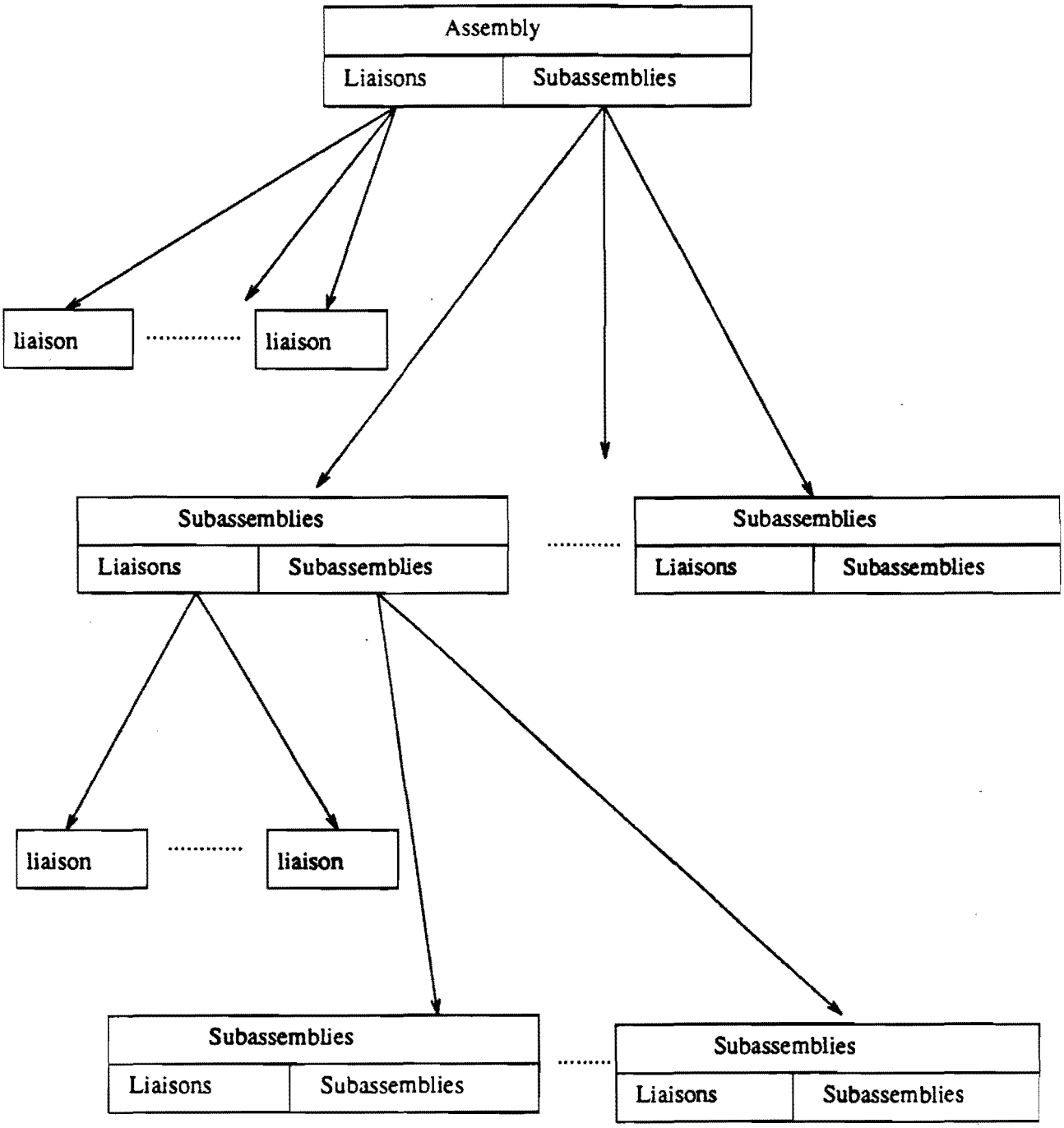


Figure 6.7. Schematic Diagram of an Assembly Hierarchy

6.2.2.3. Assembly Hierarchy Formation

With the target assembly diagram, the system forms an assembly hierarchy in two steps: (1) partition the assembly planning episodes into sets of groupings whose assembly sequences are mutually consistent, and (2) given a consistent set of groupings, construct a hierarchical structure.

This separation is based on the observation that the type of knowledge used to construct an assembly hierarchy is qualitatively different from the construction of the assembly planning episodes. The former requires reasoning about a hierarchy design in general -- SPC and SIC of subassemblies are the manifestations of the general abstraction principle. The latter requires a recall mechanism or the user input.

6.2.2.3.1. Candidate Groupings

Some of the assembly planning episodes may be inconsistent with each other because their assembly sequences are mutually contradictory: a kinematic liaison (KL-1) precedes another kinematic liaison (KL-2) by one assembly sequence and KL-2 precedes KL-1 by another assembly sequence. Using the precedence calculus (to be explained shortly), the assembly planning episodes are partitioned into consistent sets. A consistent set includes assembly planning episodes whose assembly sequences are mutually consistent. The groupings of these assembly planning episodes are called *candidate groupings*. They are used to create the subassemblies of an assembly hierarchy.

6.2.2.3.2. Hierarchy Construction Strategies

A subassembly is created from mating conditions of a grouping. "Subassemblies" field may contain children subassemblies formed by other candidate groupings. A subassembly is a hierarchical structure because it may contain other subassemblies as parts. In contrast, a grouping is a flat

structure because it may not contain other groupings as its parts.

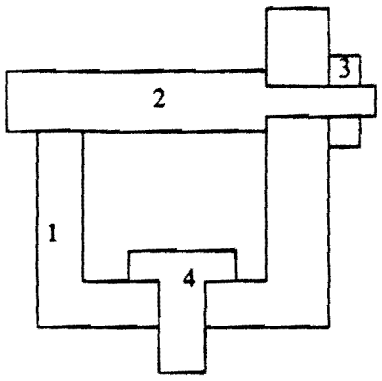
Currently, the assembly hierarchies are formed using heuristic strategies. Each strategy is illustrated by examining a case. The cases assume that the Bell-Head assembly scenario in Fig. 6.3 is resident in the memory. All the assembly planning episodes formed from Fig. 6.3 are consistent so that the candidate groupings are all the groupings instantiated from the grouping schema of Fig. 6.3.

6.2.2.3.2.1. Implicit Grouping

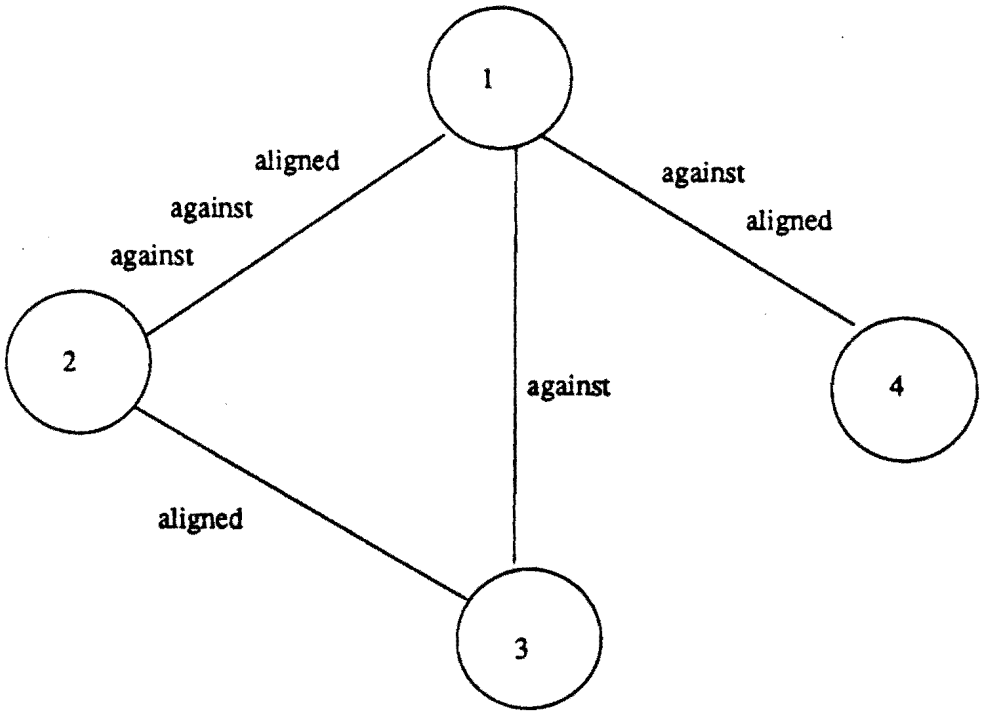
An implicit grouping occurs with the lack of complete coverage by the candidate groupings of the target assembly. Fig. 6.8 (a) shows the target assembly structure to be constructed and Fig. 6.8 (b) shows its target assembly diagram. The grouping in Fig. 6.9 (a) is formed from the Bell-Head grouping schema with the bindings: $Base to the component 2, $Sat-1 to the component 1, and $Sat-2 to the component 3. The grouping covers kinematic liaisons between the components 2 and 1 (2-1), and between the components 2 and 3 (2-3).

Using the grouping, Subassembly-1 is formed and it includes the kinematic liaisons covered by the grouping: 2-1, 2-3 and 1-3. However, the kinematic liaison between components 1 and 4 (1-4) is not covered by any grouping: it is a left-over kinematic liaison. In general, the left-over kinematic liaisons are collected into an *implicit grouping* and a subassembly is formed for the implicit grouping as well. This subassembly follows all the other subassemblies that are formed by the explicit groupings. Here, it follows Subassembly-1 by making Subassembly-1 to be its subassembly. The final assembly hierarchy is shown in Fig 6.9 (b).

Intuitively, we tend to solve those problems for which we have a solution before attacking the rest of the problem although this strategy rarely succeeds. Similarly, the implicit grouping is assembled after the subassemblies are formed by explicit groupings. Incidentally, the assembly hierarchy will produce an incorrect assembly sequence. This hierarchy will be criticized later during the

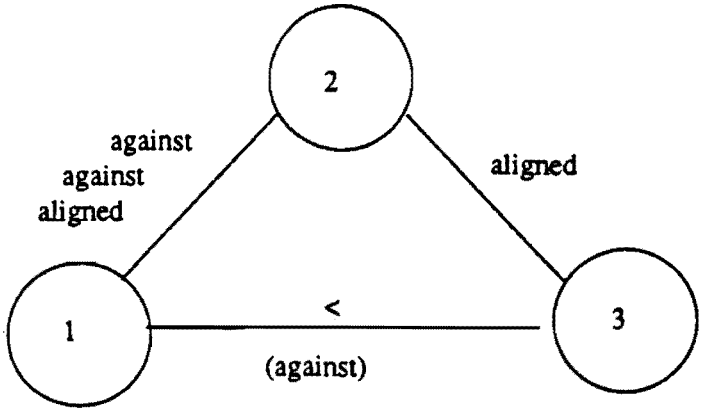


(a) Assembly Structure

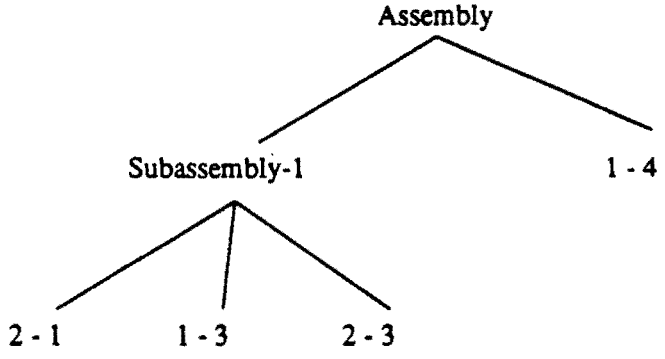


(b) Target Assembly Diagram

Figure 6.8. Assembly for Degenerate Case



(a) Assembly Planning Episode



(b) Assembly Hierarchy with the Grouping

Figure 6.9. Hierarchy Formation with an Implicit Grouping

execution phase and this grouping schema will prevented from producing a grouping for a similar target assembly diagram after the learning phase (The learning process is explained in the next chapter).

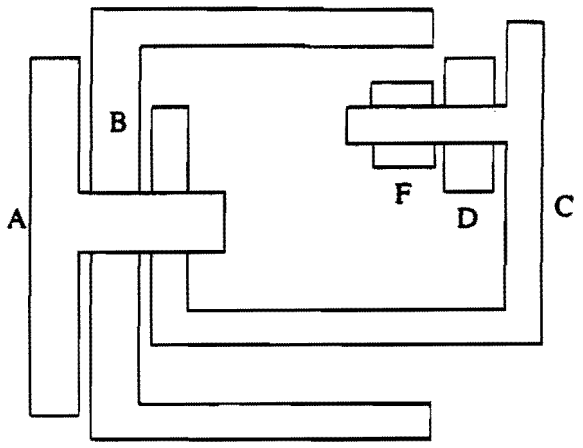
6.2.2.3.2.2. Case for Multiple Groupings

An assembly hierarchy may be organized from the interrelationships between the groupings. As an example, consider an assembly with some "twisted parts" in Fig. 6.10 (a). The groupings, Grouping-1 and Grouping-2 in Fig 6.10 (b) and (c), are formed using the Bell-Head grouping schema in Fig. 6.3. The bindings for Grouping-1 are $Base with the component A, $Sat-1 with the component B, and $Sat-2 with the component C so that Grouping-1 covers their kinematic liaisons: A-B, B-C, and A-C. The bindings for Grouping-2 are $Base with the component C, $Sat-1 with the component D, and $Sat-2 with the component F so that Grouping-2 covers C-D, D-F, and C-F.

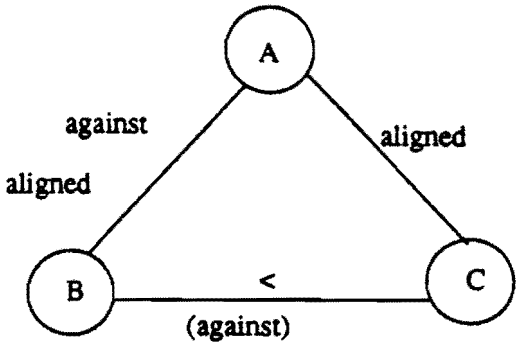
Comparing between Grouping-1 and Grouping-2, the component C is a satellite component of Grouping-1 and is a base component of Grouping-2. It would make the assembly of Grouping-2 more difficult if that component is already assembled as a satellite of Grouping-1 because the environment will be cluttered by the components of Grouping-1 when the components, D and F, are assembled to the component C. As a result, Grouping-2 should be assembled as a subassembly before Grouping-1 is assembled. Therefore, a subassembly is formed with the Grouping-2, Subassembly-1, that is made a subassembly of the assembly, formed with the Grouping-1. Subassembly-1 contains the kinematic liaisons, covered by Grouping-2 and the assembly contains those by Grouping-1. The final assembly hierarchy is shown in Fig. 6.10 (d).

6.2.3. Precedence Posting

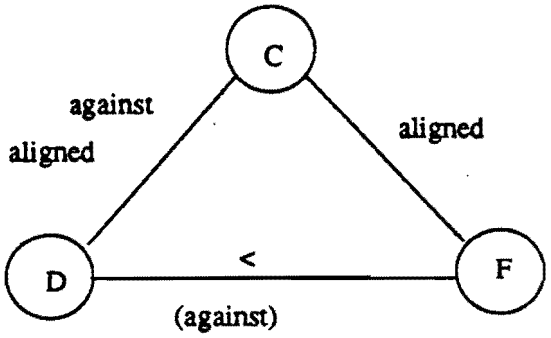
Because a precedence is a relation between events, the assembly subsequence is transformed into a sequence of assembly events by associating an event with each assembly step of the sequence. Furthermore, the ordering relations of the assembly subsequence are transformed into a conjunction of precedence relations between adjacent assembly events. For example, a sequence of A-event, B-event and C-event will be represented by a conjunction of two precedences:



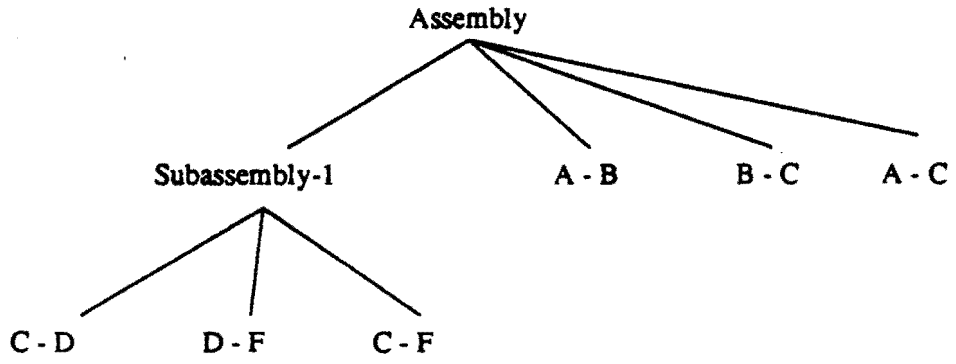
(a) Twist Assembly



(b) Grouping-1



(c) Grouping-2



(d) Assembly Hierarchy

Figure 6.10 Twist

A-event < B-event *and* B-event < C-event.

Here, "<" denotes a precedence relation. The full precedence constraint language is described shortly.

The precedences are posted at two different stages of the planning process: collecting candidate groupings and generating the final assembly sequence and monitoring the execution. After the candidate groupings and their assembly subsequences for a target assembly are formed by the grouping formation process, all the assembly subsequences are used to post the precedences. They are used to isolate sets of consistent groupings as candidate groupings whose assembly subsequences are mutually consistent. After the assembly hierarchy is formed, the precedences are posted from the assembly subsequences participating in the hierarchy as well as from SPC. Then, the posted precedences are resolved into a linear sequence of the assembly steps and the assembly sequence is posted by a conjunction of precedence relations of adjacent assembly steps for the execution monitoring.

6.2.3.1. Assembly Planning from a Precedence Specification

Alternatively, the precedences may be posted through a question/answering session [Bourjault 1985; De Fazio & Whitney, 1988]. The precedences were specified between the liaisons of the liaison diagram and boolean combinations thereof. Their main concern was two-fold: to minimize the number of questions and to generate all the viable assembly sequences from the user-provided precedences between a liaison and a boolean combinations of the liaisons. The user specified to the following two questions:

1. What liaisons must be done prior to doing *LIAISON i*?
2. What liaison must be left undone after doing *LIAISON i*?

The first question asks the user for the boolean combination of liaisons that precedes a single liaison, *LIAISON i*; the second question for the boolean combination of liaisons that follows a single liaison,

LIAISON i. Note, the boolean combination of the liaisons are allowed only either left or right hand side of the precedence relation, "<".

From the user-input of precedences, all the viable assembly sequences that were consistent with the specification were represented in a space-efficient manner using lattice-like structure. They developed a precedence calculus for the user-specified precedences. Because the precedences were provided by the user, the precedence specification was consistent. As a result, their precedence calculus mechanism fails when the precedence specification was inconsistent.

6.2.4. Precedence Calculus

On the other hand, because the precedences are posted from the empirically learned knowledge (the precedence schemas in memory) by NOMAD, they may not be mutually consistent. The precedence calculus developed here, detects inconsistencies in the precedence specification. Furthermore, the precedence constraint language allows boolean combinations of assembly events (or liaisons) to appear both left and right hand side of the precedence relation.

The precedence calculus is used at two different stages: isolating a consistent set of groupings and identifying the precedences that cause a failure when simulating the execution of the final assembly sequence. The former is carried out after the assembly sequences of the candidate groupings are posted. The latter is carried out by first posting the assembly precedences from the assembly hierarchy and detect execution failures by precedence violation (to be discussed shortly).

Each assembly step is associated with a temporal interval when the assembly step is active, an assembly event. An assembly event is represented by a temporal interval whose start time signifies when the assembly event starts and the end time signifies when the assembly event terminates. An interval is represented by a continuous line segment in the time line because the assembly event is active during the time interval without any suspension or resumption. The precedences are relations

between the assembly events.

The next section describes the precedence constraint language. The language allows boolean combinations of assembly events. The inference rules of the precedence constraint language are developed into a calculus, *the precedence calculus*.

6.2.4.1. Precedence Constraint Language

Fig. 6.11 shows two intervals whose corresponding events are labeled by "event1" and "event2". Furthermore, the bounds of the intervals are marked by their start and end time as "start" and "end" respectively. Furthermore, it shows that the end time of event1 is smaller than the start time of event2 using the "Time Line" as a metric: the event1 must precede the event2:

$$event1 < event2$$

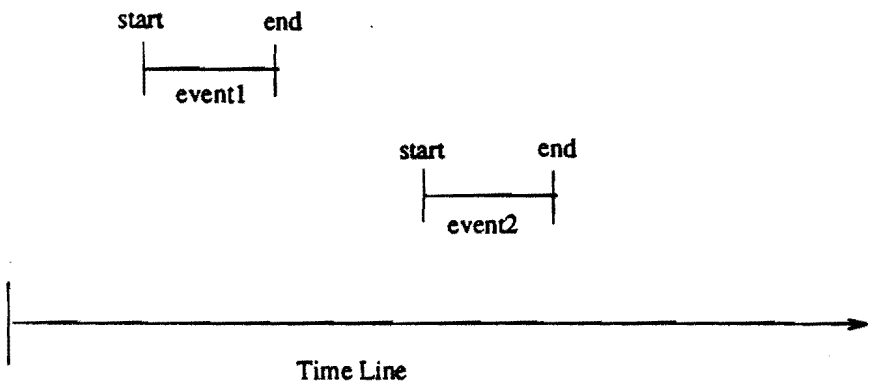


Figure 6.11. Precedence between Assembly Event1 and Event2

A precedence between two events is posted when the planner decides to wait for the completion of the event1 before initiating the event2. Furthermore, the precedences are limited to the temporal constraints between events without any overlapping intervals. As a result, the negation of a precedence is its inverse.

not (event2 < event1) <=> (event1 < event2)

That is, the event2 follows the event1 if and only if the event1 precedes the event2. The next two sections discuss the necessity of including conjunctive and disjunctive combinations of the assembly events in the precedence constraint language. For the next few sections, stability is considered as part of the assembly criteria to illustrate the need for boolean combination of assembly events in the precedence specification.

6.2.4.1.1. Precedence over the Conjunctive Events

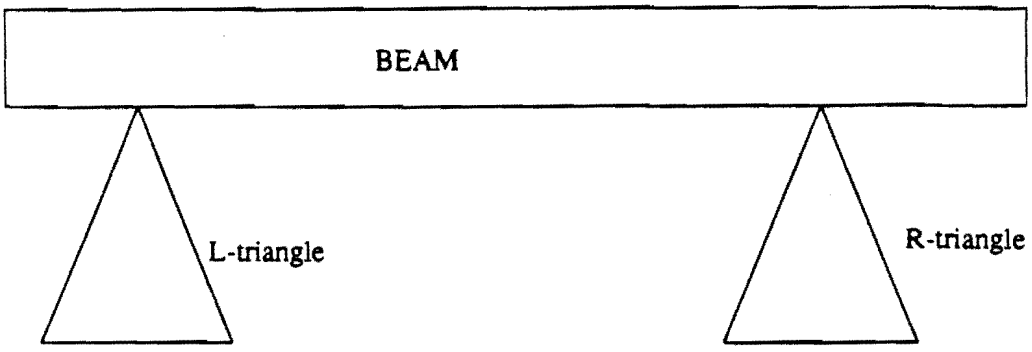
Consider an assembly in Fig. 6.12 (a) where the BEAM is balanced by two triangles, the L-triangle and the R-triangle. Both triangles should be assembled before the BEAM is assembled in order to balance the BEAM. L-event denotes the assembly event for the L-triangle; R-event denotes the event for the R-triangle; B-event denotes the event for the BEAM. Then, this precedence constraint is expressed as:

(L-event & R-event) < B-event

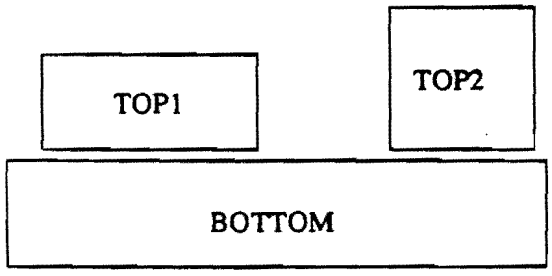
Fig 6.12 (b) shows an assembly whose BOTTOM is supporting TOP1 and TOP2 so that BOTTOM should be assembled before TOP1 and TOP2 are assembled.

B-event < (T1-event & T2-event)

where B-event for the assembly event of BOTTOM, T1-event for TOP1, and T2-event for TOP2.



(a) Conjunctive Assembly Event on the Left Hand Side



(b) Conjunctive Assembly Event on the Right Hand Side

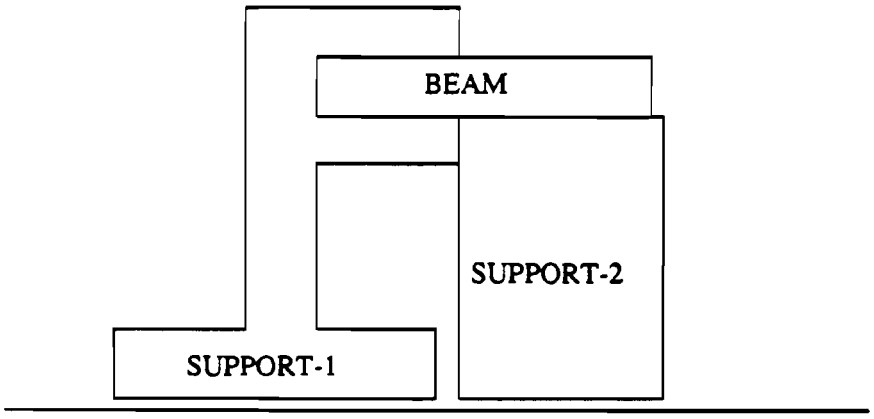
Figure 6.12. Assemblies requiring Conjunctive Assembly Events

6.2.4.1.2. Precedence over Disjunctive Events

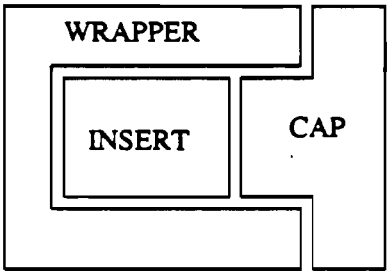
In Fig. 6.13 (a), either the SUPPORT-1 or the SUPPORT-2 are sufficient to balance the BEAM. As a result, the assembly events for the SUPPORT-1 (S1-event) or for the SUPPORT-2 (S2-event) must precede the assembly event for the BEAM (B-event):

$$(\text{S1-event} \mid \text{S2-event}) < \text{B-event}.$$

Fig 6.13 (b) shows an assembly where INSERT should be assembled either before WRAPPER or CAP:



(a) Disjunctive Assembly Event on the Left Hand Side



(b) Disjunctive Assembly Event on the Right Hand Side

Figure 6.13 Assemblies requiring Disjunctive Assembly Events

I-event < (W-event | C-event)

where I-event for the assembly event of INSERT, W-event for WRAPPER, and C-event for CAP.

6.2.4.2. Reasoning with the Precedences

This section describes inference rules for the precedence constraint language. In this section, A-event, B-event, and C-event are assembly arbitrary events.

6.2.4.2.1. Inference Rules for Conjunctive Events

When both A-event and B-event precede C-event, the last of A-event or B-event should precede C-event. However, the relative ordering between A and B events are unimportant. Furthermore, they may not overlap so that either A-event precedes B-event in the top dotted box in Fig 6.14 or B-event precedences A-event in the bottom dotted box in Fig. 6.14. However, in either case, A-event

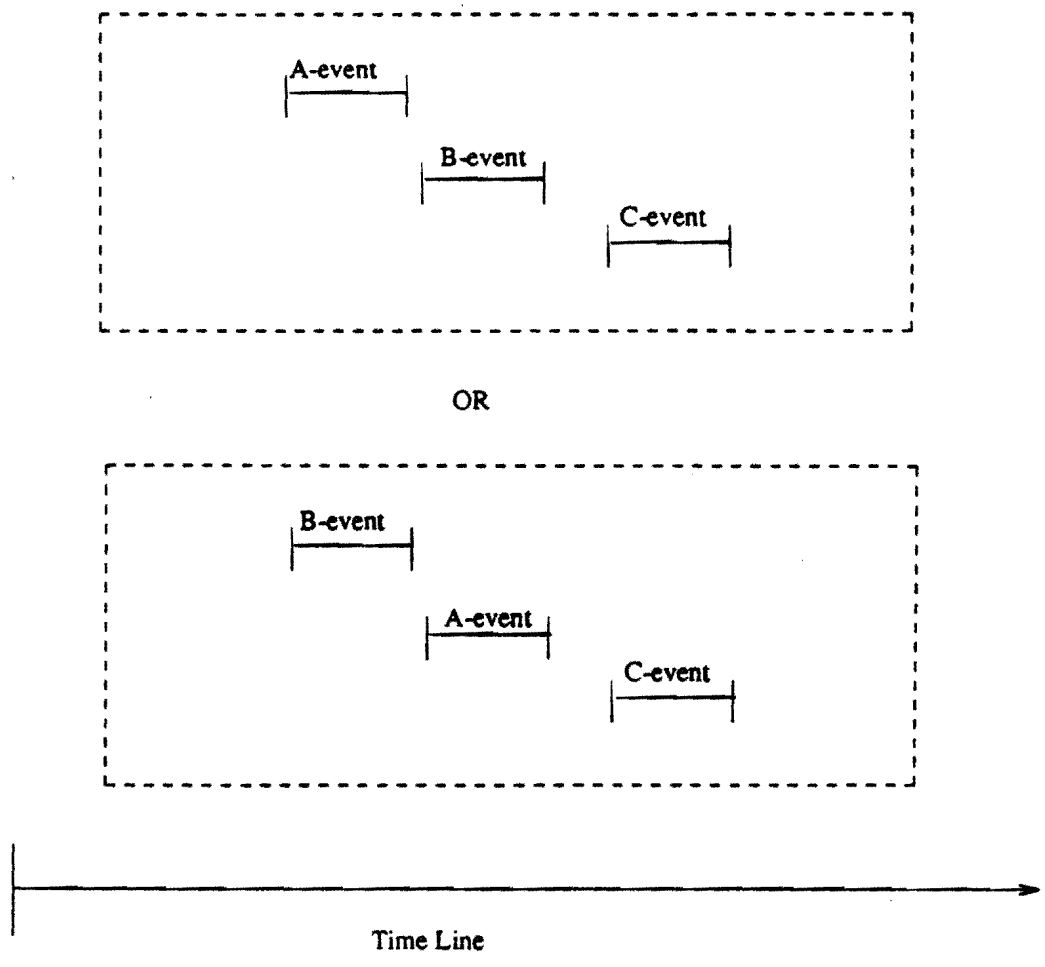


Figure 6.14. (A-event & B-event) < C-event

precedes C-event and B-event precedes C-event:

(A-event & B-event) < C-event
=>
(A-event < C-event) *and* (B-event < C-event)

Note "=>" reads "reduces to" and "&" is a combiner for the conjunctive assembly events while *and* is a boolean conjunction. Likewise, for the conjunctive assembly events on the right hand side:

C-event < (A-event & B-event)
=>
(C-event < A-event) *and* (C-event < B-event)

6.2.4.2.2. Inference Rules for the Disjunctive Events

In Fig. 6.13 (a), if the SUPPORT-1 is assembled after the BEAM, then the SUPPORT-2 must be in place to support the BEAM: if B-event precedes S1-event, then S2-event must precede B-event. Conversely, if the SUPPORT-2 is assembled after the BEAM, then the SUPPORT-1 must be in place to support the BEAM: if S2-event precedes B-event, then S1-event must precede B-event.

Consider three assembly events, A-event, B-event, and C-event, ordered by a precedence relation: A-event or B-event must precede C-event. Either A-event precedes C-event or C-event precedes A-event because there are no intersecting intervals. Suppose A-event precedes C-event. Then, no additional precedences are needed to ensure the above precedence constraint. However, if C-event precedes A-event, then B-event must precede C-event to ensure the above disjunctive precedence constraint: the bottom dotted box in Fig. 6.15. Similarly, if C-event precedes B-event, then A-event must precede C-event: the top dotted box in Fig. 6.15. In short,

(A-event | B-event) < C-event
=>
((C-event < A-event) ⊃ (B-event < C-event))
and
((C-event < B-event) ⊃ (A-event < C-event))

Here, '|' is a disjunctive combination of assembly events and ⊃ is a logical implication.

Furthermore, consider the case when the disjunction is at the right hand side: e.g., C-event must precede A-event or B-event,

C-event < (A-event | B-event).

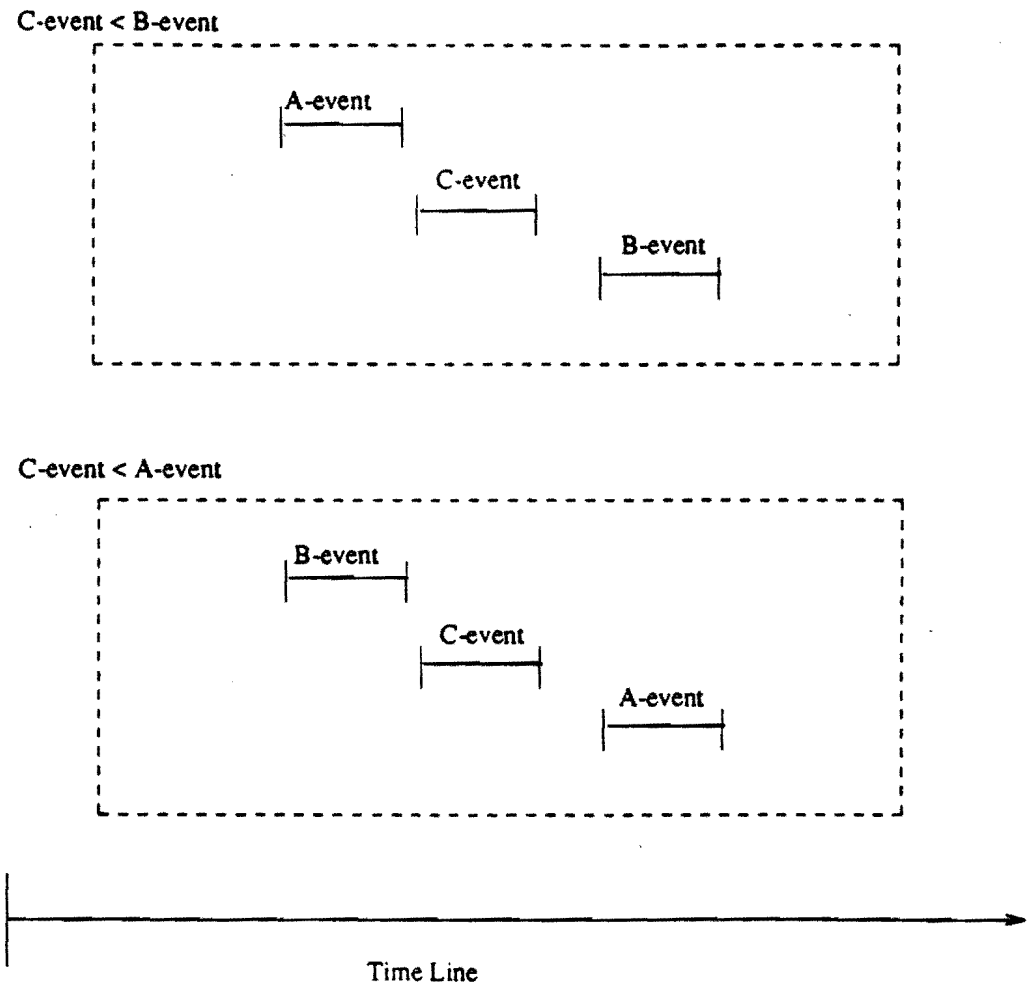


Figure 6.15. (A-event | B-event) < C-event

If A-event precedes C-event, then C-event must precede B-event; if B-event precedes C-event, then C-event must precede A-event:

C-event < (A-event | B-event)
=>
(A-event < C-event) ⊃ (C-event < B-event)
and
(B-event < C-event) ⊃ (C-event < A-event)

6.2.4.2.3. Inference Rules with Negation

When C-event may not precede both A-event and B-event, A-event or B-event must precede C-event,

not (C-event < (A-event & B-event))
=>
(A-event | B-event) < C-event

When both A-event and B-event may not precede C-event, C-event must precede A-event or B-event,

not ((A-event & B-event) < C-event)
=>
C-event < (A-event | B-event)

Thirdly, when it is not the case that either A-event or B-event may precede C-event, neither A-event nor B-event may precede C-event: that is, C-event must precede both A and B events:

not ((A-event | B-event) < C-event)
=>
C-event < (A-event & B-event)

Similarly, the last case is:

not (C-event < (A-event | B-event))
=>
(A-event & B-event) < C-event

Fig. 6.16 shows a comprehensive list of inference rules developed in this section.

Conjuntive Events

1. (A-event & B-event) < C-event
   =>
   (A-event < C-event) *and* (B-event < C-event)

2. C-event < (A-event & B-event)
   =>
   (C-event < A-event) *and* (C-event < B-event)

Disjunctive Events

3. (A-event | B-event) < C-event
   =>
   ((C-event < A-event) ⊃ (B-event < C-event))
   and
   ((C-event < B-event) ⊃ (A-event < C-event))

4. C-event < (A-event | B-event)
   =>
   (A-event < C-event) ⊃ (C-event < B-event)
   and
   (B-event < C-event) ⊃ (C-event < A-event)

Negations

5. not (event2 < event1) <=> (event1 < event2)

6. not (C-event < (A-event & B-event))
   =>
   (A-event | B-event) < C-event

7. not ((A-event & B-event) < C-event)
   =>
   C-event < (A-event | B-event)

8. not ((A-event | B-event) < C-event)
   =>
   C-event < (A-event & B-event)

9. not (C-event < (A-event | B-event))
   =>
   (A-event & B-event) < C-event

Figure 6.16 Comprehensive List of Precedence Inference Rules

6.2.4.2.3.1. Analysis of Bourjault's Approach

In Bourjault's original work [De Fazio & Whitney, 1988], the precedences were specified by asking the user the following two yes-no questions with a negation:

1. Is it true that L_i cannot be done after L_j and L_k have been done?
2. Is it true that L_i cannot be done if L_j and L_k are still undone?

L_i, L_j, and L_k are liaisons in the liaison diagram. Equivalently, these liaisons are assembly events in the precedence calculus. The *first question* is asking the truth value of:

$$\text{not}\,((L_j \,\&\, L_k) < L_i)$$

Pushing the negation inwards, the precedence relations is reduced to:

$$L_i < (L_j | L_k)$$

The *second question* is asking the truth value of:

$$\text{not}\,(L_i < (L_j | L_k))$$

Pushing the negation inwards, the precedence relation is reduced to:

$$(L_j \,\&\, L_k) < L_i$$

6.2.5. Executing the Final Assembly Sequence

From the assembly hierarchy, the final assembly sequence is generated as a sequence of the assembly steps. Each assembly step is carried out by a mating operation whose *postconditions* are the mating conditions between the geometric features of the fixed component (a base component of some grouping, used in the assembly hierarchy) and those of a moving component (a satellite component of the grouping). The mating operator is executed by SRE that computes a graphic translations and rotations of the satellite component.

The position map is used to represent the scene so that the assembly instruction updates the position map for the new scene after the assembly step. Multiple position maps can be maintained

efficiently because only the base frames of all the objects of the scene have to be maintained for the scene.

6.2.5.1. Execution Monitoring

The user has to alert the monitor during the assembly if any of the mating operation fails because there are no collision free paths for the mating object to its final assembled position. Currently, the user inspects which components of the already constructed subassembly are causing the blockage. Then, he contradicts the existing precedence relations of the current assembly sequence that caused assembly steps to be planned prematurely so that their objects are prematurely assembled. That is, the execution failures are monitored by the user and are simulated by the precedence violations of the assembly sequence.

For example, in assembling the Bell-Head of Fig. 6.1, if the RING is assembled to the PIN before the HEAD, the HEAD can not be assembled to the PIN because the PIN and the RING together block any collision free paths as shown in Fig. 6.2: HEAD should be assembled before either the PIN or the RING. Because the PIN is the base component, assembling the HEAD should precede assembling the RING. Therefore, the assembly planning, based on the collision physics, predicts this impasse by envisioning the failure and avoids it during the planning. In the empirical assembly planning approach, the planner does not plan by envisioning the collisions but rather from previously learned schemas for the groupings and the assembly sequences. As a result, the execution allows the system to backtrack on previous decisions and avoid the impasse when encountered with "similar" target assembly.

Eventually, GMS should be equipped with a trajectory planner that would detect obstacles in the scene during an assembly operation. Then, using the partially executed assembly procedure, it may assert these additional precedence relations autonomously when no collision free paths exist.

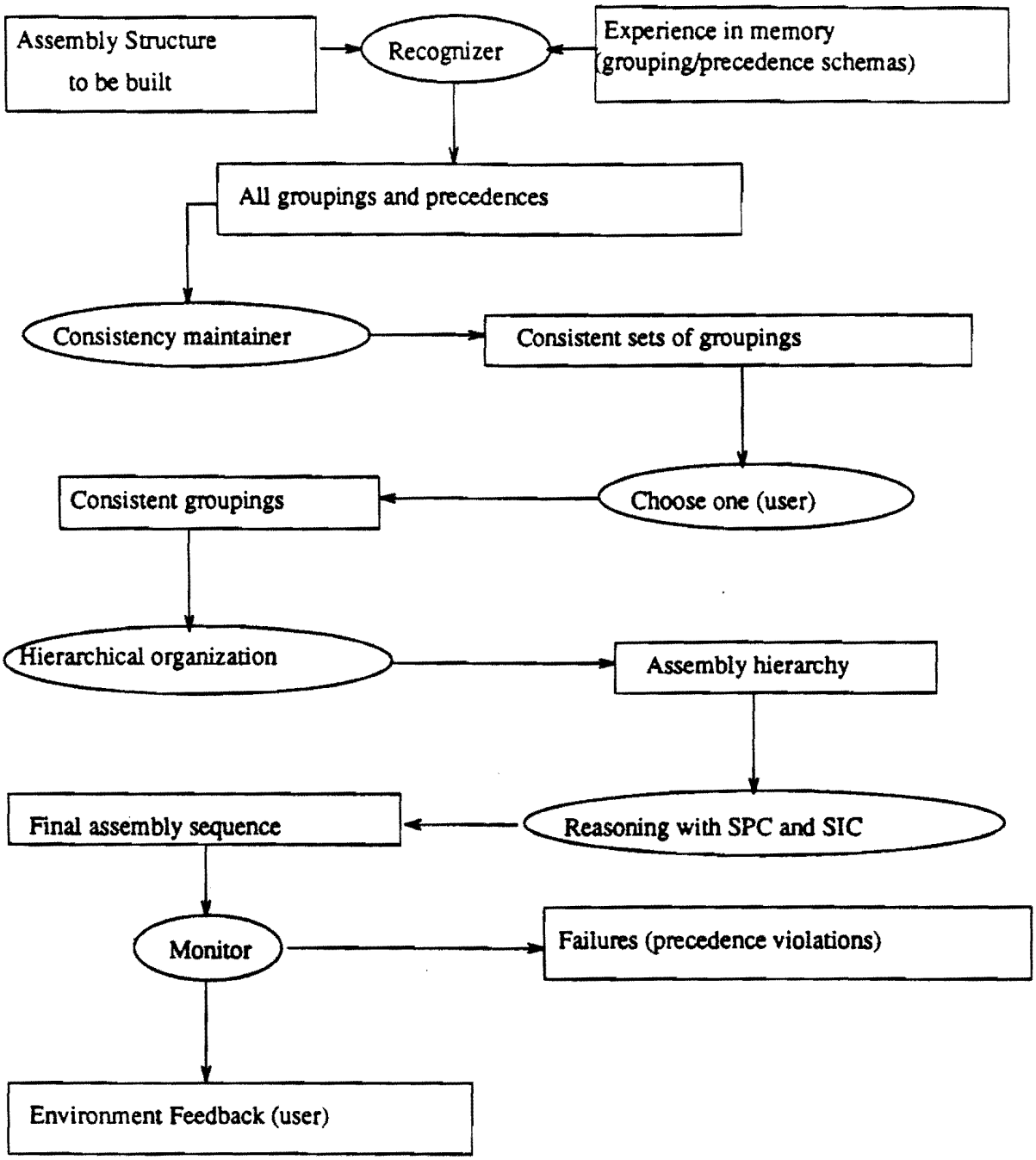


Figure 6.17. Overall Planning and Execution Processes

6.2.6. Summary of Empirical Assembly Planning/Execution

The assembly planning, execution and user interactions are summarized in a flowchart format in Fig. 6.17. The retangles are data and the ellipses are procedures. The user interactions are marked by "user" in parentheses.

CHAPTER 7.

EMPIRICAL MACRO-OPERATOR SCHEMA LEARNING

In Chapter 5, the collision physics, the first principle of the environment, was used to project the effects of planning assembly steps. The planner used the collision physics as a task theory. Unfortunately, reasoning from the collision physics as an assembly task theory was intractable. In order to plan for mechanical assemblies in the physical world, the planner requires the entire Newtonian mechanics in addition to collision physics. Therefore, the planning task is even more intractable.

However, people as intelligent planners could solve the mechanical assembly problems long before Issac Newton! Hence, an intelligent planner may not require the first principles of the physical world to deal with the mechanical assembly task domain. Hence, humans must be, at least initially, *empirical planners.*

An empirical assembly planner reasons from an empirical assembly task theory that is described in terms of the observable features. The theory is used directly by the assembly planner: the problem of assembling the target assembly is decomposed into the subproblems of assembling its groupings and to suggest their correct solutions as assembly plan segments of the final assembly sequence. Therefore, large portion of the planning task becomes a perceptual task requiring less reasoning than the projection-based planners.

However, the empirical assembly task theory is necessarily *approximate* because the theory is formed from empirical observations inductively. Reasoning from an empirical assembly task theory is trading the soundness and completeness of reasoning from the first principle with the tractability of the empirical theory. The learning task of an empirical assembly planner is *not* to discover a *causal theory* but rather to formulate an adequate enough task theory that can be used to plan for the scope of

assembly task situations to be encountered.

Previously, planning systems formulated the task theory as macro-operator schemas. These systems generalized a planning episode into a macro-operator schema carefully so that the schema is guaranteed to be correct. The generalization was deductive -- *analytic macro-operator schema learning*, described in the next section.

On the other hand, the empirical assembly planner accumulates the assembly planning episodes as macro-operator schemas as an empirical assembly task theory. They are formed by an *inductive* generalization of the planning episodes, *emprical macro-operator schema learning*.

7.1. Analytic Macro-operator Schema Learning

Previous systems [Fikes & Nilsson, 1972; Segre, 1987][1] that learned from planning episodes used a planning model based on the state-space model of problem-solving. The planning model had three main ingredients: operator schemas, a world state description, and a goal state recognizer. Then, a planning episode was generated,

(1) by instantiating the operator schemas with respect to the world state as operators and then, applying each operator in some sequence to transform the world state until a goal state is generated and recognized or

(2) by the user who may instruct the sequence of operators.

A macro-operator schema is a generalized description of the planning episode and is stored as a planning experience. With macro-operator schemas, a planner may avoid generating a sequence of primitive operators when it can be generated by a *single application* of the macro-operator schema in

[1]Although ARMS system by Segre is not a planning system but rather an understanding system, it had all the ingredients of a planning system.

memory. The sequence of operators is an instance of the macro-operator schema called a macro-operator. The macro-operator may shorten the search process when it forms a part of the final plan.

Previously, a macro-operator schema was learned by a deductive generalization of a planning episode, explanation-based learning (EBL) [Mitchell & Keller & Kedar-Cabelli, 1986; De Jong & Mooney, 1986]. The constants of a plan episode was generalized minimally so that the operator schemas together with domain specific knowledge logically implies the macro-operator schema after the generalization. The generalization was deductive; hence, the learning is called *analytic macro-operator schema learning*.

7.2. Empirical Macro-operator Schema Learning

NOMAD accumulates the planning episodes in its macro-operator schemas as planning experiences like STRIPS and ARMS. Unlike them, macro-operator schemas are formed by inductively generalizing planning episodes. As a result, using the macro-operator schemas, the assembly planner may construct an incorrect assembly sequence. Monitoring the execution of an assembly sequence empirically validates or invalidates the macro-operator schemas used to generate segments of the sequence.

In Chapter 6, NOMAD, as a domain-dependent planner, names the elements of the macro-operator learning systems specially. A precedence schema is comparable to a macro-operator schema. An assembly sequence of the precedence schema is comparable to a macro-operator. A grouping schema represents the effect list of a macro-operator schema. There is no comparable structure for the precondition of the macro-operator schema because the only operator of NOMAD is the mating operator and the mating operator is always executable. It may fail if there is no collision-free trajectory for accessing a volume of space where to achieve the desired configuration. This condition constrains the operator *during* its operation rather than before.

In the planning model in the previous section, the plans are generated using a search mechanism in the space of world states to explore alternative concatenations of the operators to reach the goal state. In comparison, NOMAD constructs plans from a memory instantiations and an organization of the instances as described in the previous chapter. To restate the planning process in terms of the macro-operator learning systems, NOMAD

(1) instantiates the macro-operator schemas,

(2) combines the macro-operators by organizing them into a hierarchy, and

(3) generates the final plan from the hierarchy.

The next section will describe the related researches in machine learning. The rest of the sections will describe how NOMAD learns the grouping and precedence schemas from assembly planning episodes.

7.2.1. Structural Concept Learning

Learning methods based on an empirical generalization require comparing objects as instances or non-instances of a concept to uncover similarities and dissimilarities. A plan structure is a highly *structured object* consisting of diverse kinds of operators and temporal constraints between them. In general, a structured object consists of component objects and relations among them like an assembly structure or a plan instance.

When comparing two structured objects, a component object from one structured object corresponds with one from the other structured object, known as the *object-correspondence problem.* Suppose there are M distinct objects in each of two instances; there are M! possible object correspondences between them, assuming one-to-one mapping (the complexity is worse otherwise). Therefore, comparing two structured objects is an inherently explosive process. To compound the difficulties,

many instances should be compared to uncover regularities.

On the other hand, when an object is described purely in terms of its attributes, the inner-structure of the object is abstracted out. Hence, comparing many instances described in terms of attributes is feasible because it requires no mechanism for the object-correspondence problem. Most machine learning research in empirical generalization learned from the attribute-based training instances.

This simplifying assumption allowed researchers to concentrated on the generalization mechanisms without worrying about the object-correspondence problem. The main learning task here is to find a *plausible* generalization covering all the positive instance of a concept but none of the negative instances of the concept. This effort contributed to the development of many useful symbolic generalization techniques [Quinlain, 1983; Chen, 1988] and a theory of generalization as an inference process [Michalski, 1983].

Nonetheless, it has been a long standing research problem in machine learning to learn from structured objects. So far, learning from spatially structured objects [Winston, 1970; Larson, 1977; Stepp, 1984] and from temporally structured objects [Michalski & Ko & Chen, 1987; Chen, 1988] have been studied independently. An assembly structure is a spatially structured object; an assembly sequence is a temporally structured object. Learning the macro-operator schema involves learning from the groupings and assembly sequence segments: the former are spatially structured objects and the latter are temporally structured objects. Therefore, it is necessary for NOMAD to be able to learn from both spatially and temporally structured objects.

Furthermore, these systems were designed for answering *how* to learn structural concepts but leave the task of determining *when* and *what* concept to learn to the user. This learning paradigm by itself is inadequate for learning tasks involving planning experiences. When the learned concepts are used and does not cause any failure, the concepts are empirically validated to the extent that the

concepts are used so far. On the other hand, when the learned concepts are used and cause a failure, the concepts should be eliminated or changed. The "when" question should be answered in some way for any "incremental" learning systems from positive and negative examples including some of the above systems [Winston, 1970; Chen, 1988]

Determining what concept to learn becomes important for learning from planning experiences because a planner invokes multiple concepts to construct a plan. The planner should maintain which concepts are involved in each planning step so that when a failure occurs, only some of the concepts used by the planner are responsible for the failure. In short, a system that learns from planning experience should determine *when* and *what* concept to learn as well as *how* to learn the concept. The next two sections reviews the previous works in the empirical learning from structured objects.

7.2.1.1. Empirical Learning from the Spatially Structured Objects

Winston's seminal work [Winston, 1970] developed a domain-independent method for learning structural concepts from the visual scene. It divided the objects of the scene into groups based on observed similarities and dissimilarities, a grouping procedure. Each grouping as a structured object is compared with other structured object as instances or non-instances of a structural concept, a comparison procedure.

The grouping procedure was later improved by CLUSTER [Stepp,. 1984]. It developed a number of useful similarity metrics for the clustering criteria and heuristic control algorithm for the hierarchical cluster formation. The comparison procedure was improved by INDUCE [Larson, 1977]. It introduced an explicit two-layered approach to the object comparison where the structural relations (links) were matched before the object attributes were compared.

The comparison procedure used a network matching algorithm for the subgraph isomorphism problem: each structured object is represented by a labeled graph and multiple structured objects are

merged by common subgraphs from each graphical representation of the objects. These systems were applied to the recognition of *spatial* structures. None of the systems by themselves would be able to learn interesting macro-operators in the assembly task domain if they are applied without any further processing.

7.2.1.2. Empirical Learning from Temporally Structured Objects

Learning systems above handle sequences like any other structures ignoring the temporal constraints in learning such structures. Recently, learning from temporally structured objects [Michalski & Ko & Chen, 1987; Chen, 1988] was studied as a separate learning problem.

In SPARC[2] [Michalski & Ko & Chen, 1987], the learning task was to induce a pattern from a sequence of objects or events, described by multiple attributes. The pattern can explain a sequence including the future events. The generalization mechanism used the syntactic constraint of the types of rules. Each different syntactic constraint was categorized as rule models. For a given rule model, the input observation was transformed specially before the inductive generalization was applied to parts of the transformed data.

Each object is a *part* of an interdependent structured object and the task is to hypothesize the whole object from the parts. This learning required a "part-to-whole" generalization mechanism. The *interrelationships* between parts themselves describe the concept.

On the other hand, the structural concept learning systems in the previous section were given each training instance as an *independent* member or a non-member of a concept. They required an "instance-to-class" generalization[3].

---

[2]Sequential PAttern ReCognition

[3]Class is synonymous with concept.

In "instance-to-class" generalization, the observed differences were used indirectly in the concept formation: it was used to modify the concept description but they themselves were not part of the concept description. For example, Winston's system modified the object model incrementally from the observed difference between training instances with emphatic relations like must and must-not links.

Like the learning systems in the previous section, neither SPARC nor TIM[4] can learn macro-operators for the empirical assembly planner because it requires mechanism for learning from both spatially and temporally structured objects.

7.2.1.3. Object-correspondences between Macro-operators

In general, a robot plan may consist of grasping, positioning, and mating operations. A grasping operation in one plan structure may not correspond to a positioning operation in another plan structure. When a plan structure contains diverse kinds of operators, it is cumbersome to compare multiple macro-operators or any of their substructures because the comparison may require additional transformations in order to make the comparison between *homogeneous* structures.

An assembly planning episode of NOMAD is represented homogeneously as a sequence of mating operations. It is sufficient to represent an assembly sequence for a grouping as a sequence of its satellites. Comparing multiple assembly sequences for a grouping is equivalent to comparing multiple sequences of its satellite components.

A component is represented by vertices, edges, and faces. A component as part of an assembly structure is considered as a primitive object whose inner structures are abstracted out. As a result, the comparison between assembly sequences with respect to a grouping is reduced sequences of attribute-based representation of the satellite components.

[4]Time-based Inductive Machine [Chen, 1988]

7.2.1.3.1. Levels of Representation

The representation of an object as a structured entity or a primitive object depends on the *levels of abstraction* that are required for a particular inference task. These levels of abstraction result in a hierarchical structure.

An assembly hierarchy represents levels of representation of goals for the assembly planner. The children subassemblies in the assembly hierarchy represent single entities with respect to the parent subassembly in deriving the assembly sequence. Because the assumptions behind forming each subassembly as an indivisible single entity may fail, SPC and SIC are used to detect such failures and correct the responsible schema used to form such an abstraction.

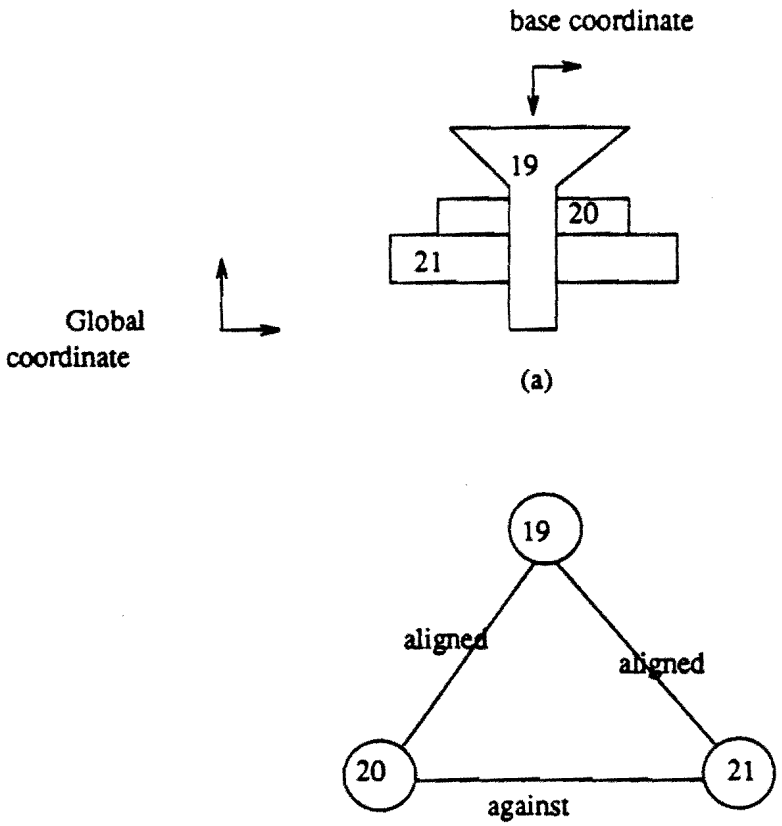
7.2.2. Representation and Processing of Internal Structures

This section introduces the internal structures of the diagram and informal representations of the various constructs of the empirical assembly planner in the previous chapter. Furthermore, this section follow the sequence of the planning process of the previous chapter: target assembly, grouping and assembly sequence segments, assembly hierarchy formations. Then, a schema formation procedure is described.

7.2.2.1. Target Assembly

Fig. 7.1 (a) shows WEDGE assembly as a target assembly and the target diagram in Fig. 7.2 (b). Note, a circle denotes a component and its label denotes the part number: e.g., the circle with 19 inside denotes [PART19]. The target assembly is represented internally as a field and a value pairs. The field names and their meanings are as follows:

NAME: *name of the assembly*
PARTS: *components that are part of the assembly*



(b) Target Assembly Diagram

Figure 7.1. WEDGE Assembly

LIAISONS: *kinematic liaisons of the target assembly diagram*
HIERARCHIES: *alternative assembly hierarchies for the assembly*

LIAISONS contain the kinematic liaisons between parts of the target assembly diagram. Liaison([PART19],[PART20]) contains all the mating conditions between geometric features of [PART19] and [PART20] -- an aligned condition between the center axis of [PART19] and the center axis of the hole of [PART20]. Fig. 7.2 shows the WEDGE assembly. A1 in HIERARCHIES field is formed as the root assembly hierarchy after the groupings and assembly sequences are determined.

NAME: WEDGE
PARTS: ([PART19] [PART20] [PART21])
LIAISONS:(Liaison([PART20],[PART21]) Liaison([PART19],[PART21]) Liaison([PART19],[PART20]))
HIERARCHIES: (A1)

Figure 7.2. Internal Structure of WEDGE assembly

7.2.2.2. Grouping and its Assembly Sequence Segment

When there are no relevant schemas in memory to the given target assembly, the system asks the user for the groupings and their assembly sequence segments: a sequence of satellites for each grouping. An internal structure of a grouping is as follows:

INDEX: *index to distinguish the grouping from others*
GSCHEMAS: *grouping schemas subsuming this grouping*
BASE: *base component*
SATELLITES: *list of satellite components*
LIAISONS: *list of kinematic liaisons between the base and satellites*
PINSTS: *alternative assembly sequences used to order the satellites*

For the target assembly of Fig. 7.2, a grouping is formed by the user: [PART19] as a base component and [PART20] and [PART21] as satellites. Fig. 7.3 shows internal structure of the grouping, GI-1. GSCHEMA-1 is a grouping schema. It is generated when the grouping's assembly sequence segment, PI-1, is used to plan a successful assembly plan and GI-1 is missing a grouping schema which is the case here because the user created it. All the kinematic liaisons between the base and satellite components are copied from those specified by the target assembly diagram: Liaison([PART19],[PART20) and Liaison([PART19],[PART21]).

When a grouping is specified by the user, its assembly sequence is also specified by the user. An internal structure of an assembly sequence for a grouping is as follows:

INDEX: *index to distinguish the grouping from others*
PSCHEMAS: *precedence schemas subsuming this assembly sequence*
GINST: *a grouping for which this is an assembly sequence*
SEQUENCE: *sequence of satellites*

INDEX: 1
GSCHEMAS: (GSCHEMA-1)
BASE: [PART19]
SATELLITES: ([PART20] [PART21])
LIAISONS: (Liaison([PART19],[PART20]) Liaison([PART19],[PART21]))
PINSTS: (PI-1)

Figure 7.3. Grouping Structure: GI-1

DIFF-SEQUENCE: *the symbolic derivative of SEQUENCE as a sequence of difference of adjacent objects*
LIAISONS: *kinematic liaisons between satellites*

For GI-1, the user specifies to assemble [PART20] before [PART21]. The sequence is internalized into PI-1. Fig. 7.4 shows the internal structure of PI-1. Diff-part([PART20],[PART21]) contains all the difference of attributes between [PART20] and [PART21].

In the previous chapter, a pair of a grouping and its assembly sequence is represented together in a diagram as an assembly episode. Fig. 7.5 shows the diagram representation corresponding to GI-1 and PI-1 together above.

7.2.2.3. Assembly Hierarchy

After groupings and assembly sequences are determined, they are partitioned into maximally consistent sets of groupings whose assembly sequences are mutually consistent is determined. From

INDEX: 1
PSCHEMAS: (PSCHEMA-2 PSCHEMA-1)
GINST: GI-1
SEQUENCE: ([PART20] [PART21])
DIFF-SEQUENCE: (Diff-part([PART20],[PART21])))
LIAISONS: (Liaison([PART20],[PART21]))

Figure 7.4. Assembly Sequence for a Grouping: PI-1

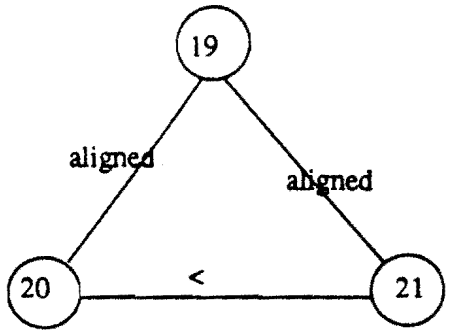


Figure 7.5 Assembly Episode Diagram for WEDGE Assembly

a consistent set of groupings, an assembly hierarchy is formed. The internal structure of an assembly hierarchy is as follows:

INDEX: *index to distinguish itself from other assembly hierarchies*
GINST: *a grouping used to create this assembly hierarchy*
BASE: *a base component usually not used except for implicit grouping*
SATELLITES: *also, for implicit grouping*
SUBHIERARCHIES: *children subassembly hierarchies*

Liaisons field of Fig. 6.7 in the previous chapter is implicitly represented inside the grouping of GINST. Subassemblies field is SUBHIERARCHIES.

For the TRIANGLE assembly as a target assembly, GI-1 covers all the components and no alternative groupings are formed. Therefore, the hierarchy formation procedure bypasses the determination of the consistent set of groupings. Using GI-1, the assembly hierarchy is formed with depth 0, A1, that is shown in Fig. 7.6.

INDEX: 1
GINST: GI-1
BASE: [PART19]
SATELLITES: ([PART20] [PART21])
SUBHIERARCHIES: ()

Figure 7.6 Assembly Hierarchy: A1

7.2.2.4. Support Structure Embedded in the Assembly Sequence

After the hierarchy is formed, the planner posts the precedence relations of the final assembly sequence with the support structure. A precedence relation of the final assembly sequence may be asserted by instances of the schemas in memory, the subassembly precedence constraint (SPC) or indirectly by the user who forms groupings and their assembly sequences. The user formed GI-1 and PI-1 in the ongoing example.

A grouping and SPC together may support a precedence relation between assembly steps of a subassembly and those of the parent assembly when the grouping is used to create the subassembly. An assembly sequence segment posted by a precedence schema supports all the precedence relations derived from the sequence segment. The user supports a precedence relation indirectly by creating a grouping and its assembly sequence. Additional precedence relations are derived using the precedence calculus.

Continuing with the WEDGE assembly as an example, Fig. 7.7 shows the support structure. The precedence relation from PI-1, its source of support, is all the precedence relation in the final assembly sequence:

[[PART20],[PART19]] < [[PART21],[PART19]].

In general, all the precedence relations of assembly sequence are asserted as statements into an assertional database. Using the inference rules of the precedence calculus, the support structure for pre-

cedence relations are maintained.

7.2.2.5. Monitoring an Execution Failure

As described in the previous chapter, an execution failure is simulated by a precedence violation. The execution monitor reports a failure to mate a component by asserting a precedence relation that contradicts a precedence relation of the final assembly sequence that is responsible for blocking the collision-free path of the mating component.

With the precedence violation, the system examines the sources supporting the violated precedence relation of the assembly sequence. Then, it generates a culprit list, a list of candidate instances of the schemas in memory whose application is responsible for the failure. The credit-assignment of the responsible support is domain dependent and will be described with individual learning scenarios shortly.

7.2.2.6. Formation or Updates of the Schemas

Note, NOMAD does not attempt to replan right away when the assembly sequence fails. By monitoring the execution of an assembly sequence, the empirical assembly planner accumulates the evidence for or against the schemas used to create the segments of the sequence. In response, the

Figure 7.7 Support Structure for WEDGE assembly

schemas are updated with plan failures. However, because the schemas are modified and updated by assimilating the planning episodes as experiences, it does not repeat the same failure. Hence, it indirectly corrects its mistake.

After monitoring the execution of the final assembly sequence, the groupings and assembly sequence segments used to create the assembly sequence are assigned whether they are positive or negative instances of some schema in memory. If there are no schemas associated with an instance, then it is created by the user and new schemas is formed from the instance.

In the ongoing WEDGE assembly example, the assembly sequence succeeds. PI-1 is a positive instance of some precedence schemas that not yet exist. GI-1 is also a positive instance because its only assembly sequence is a positive instance. Because they are formed by the user and missing schemas, new schemas are created from the instance.

7.2.2.6.1. Grouping Schema

An internal structure of a grouping schema is as follows:

INDEX: *index numbers grouping schemas uniquely*
BASE: *base component schema*
SATELLITES: *satellite component schemas*
LIAISONS: *liaison schemas*
GINSTS: *pairs of a grouping and the bindings of schema variables and their instances in the grouping*
POS-GINSTS: *positive groupings*
NEG-GINSTS: *negative groupings*
PSCHEMAS: *precedence schemas that are used to order satellite schemas*

Fig. 7.8 shows the new grouping schema formed for GI-1, GSCHEMA-1. $ denotes a variable. $part1, $part2, and $part3 are variables denoting component schemas. $Liaison1 and $Liaison are variables denoting liaison schemas: a generalized description of a kinematic liaison. Currently, only one instance is collected for GSCHEMA-1, GI-1. GINSTS contain the binding list of GI-1: a variable of the grouping schema and with an instance in the grouping. For example, $part1 of GSCHEMA-1 is bound to [PART-19] of GI-1. POS-GINSTS and NEG-GINSTS collect positive and negative

INDEX: 1
BASE: $part1
SATELLITES: ($part2 $part1)
LIAISONS: ($Liaison2 $Liaison1)
GINSTS: ((GI-1 ($part1 . [PART19]) ($part3 . [PART20])
($part2 . [PART21])($Liaison1 .Liaison([PART19],[PART21]))
($Liaison2 . Liaison([PART19],[PART20])))
POS-GINSTS: (GI-1)
NEG-GINSTS: ()
PSCHEMAS: (PSCHEMA-2 PSCHEMA-1)

Figure 7.8 Grouping Schema for GI-1

instances of the GSCHEMA-1 that are determined by the execution monitor. PSCHEMAS field contains a list of alternative precedence schemas used to order the satellite schemas of the grouping schema like PINSTS field for GINST.

7.2.2.6.2. Precedence Schema

Internal structure of a precedence schema is as follows:

INDEX: *index that uniquely identifies the precedence schema*
GSCHEMA: *grouping schema for which the precedence schema is used to order the satellite schemas*
SEQUENCE: *sequence of satellite component schemas*
DIFF-SEQUENCE: *symbolic differential of the sequence*
POS-PINSTS: *positive instances of the pschema*
NEG-PINSTS: *negative instances of the pschema*

POS-PINSTS and NEG-PINSTS are positive and negative assembly sequences or instances of a precedence schema. PI-1 formed by the user is a positive instance of both precedence schemas. Using PI-1, two alternative precedence schemas are formed, PSCHEMA-1 and PSCHEMA-2. Their only difference is between $Diff-part1 and $Diff-part2. They are alternative generalizations of Diff-part([PART20],[PART21]) which is described by a conjunction of differential selectors [Michalski, Ko & Chen, 1987]:

INDEX: 2
GSCHEMA: GSCHEMA-1
SEQUENCE: ($part3 $part2)
DIFF-SEQUENCE: ($Diff-part2)
POS-PINSTS: (PI-1)
NEG-PINSTS: ()
(a) PSCHEMA-2

INDEX: 1
GSCHEMA: GSCHEMA-1
SEQUENCE: ($part3 $part2)
DIFF-SEQUENCE: ($Diff-part1)
POS-PINSTS: (PI-1)
NEG-PINSTS: ()
(b) PSCHEMA-1

Figure 7.9 Two Precedence Schemas from PI-1

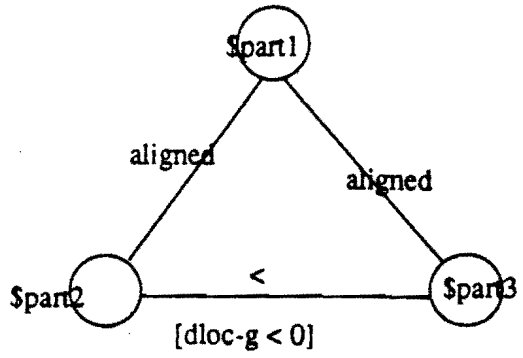
[dloc-b([PART20],[PART21]) < 0][dloc-g([PART20],[PART21]) > 0].

The differential selectors are derived from individual locations of the parts in global and base coordinates. Dloc-b computes the difference between locations with respect to the base coordinate and dloc-g with respect to the global coordinate, the coordinates are shown in Fig. 7.1. Alternative generalizations of Diff-part([PART20],[PART21]) by dropping condition [Michalski, 1983] are [dloc-b($part3,$part2) < 0] for $Diff-part2 and [dloc-g($part3,$part2) > 0] for $Diff-part1.

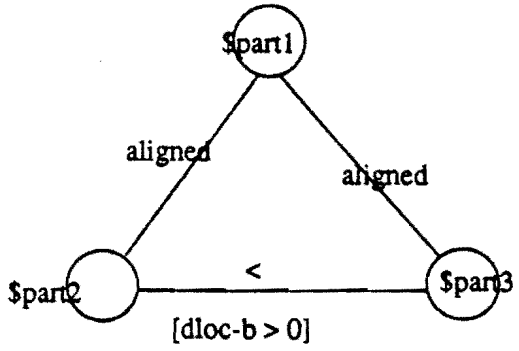
In the previous chapter, a pair of a grouping and a precedence schemas is represented together in a diagram as an assembly scenario. Fig. 7.10 (a) shows the diagram representation corresponding to GSCHEMA-1 and PSCHEMA-1 and Fig. 7.10 (b) for GSCHEMA-1 and PSCHEMA-2.

7.2.3. Refinement of Macro-operator Schema Library upon Failures

Because macro-operator schemas are formed empirically, it should be monitored whether they are used to create faulty assembly sequences. When a precedence violation is detected, the system



(a) GSCHEMA-1 & PSCHEMA-1



(b) GSCHEMA-1 & PSCHEMA-2

Figure 7.10 Assembly Scenarios

traces back the groupings and assembly sequence segments supporting the violated precedence relation of the final assembly sequence, using the support structure of the final assembly sequence. The identified instances are tagged as negative instances of the associated schema. The next two sections describe these credit-assignment processes and learning to apply empirically validated schemas over those whose instances contributed to the failure.

When the violated precedence comes from an assembly subsequence, the source of the failure is in applying the associated precedence schema. When the violated precedence comes from SPC, the grouping associated with the subassembly is blamed when the assembly hierarchy is constructed from only one grouping. The current implementation can not handle cases where more than one groupings are involved in constructing the assembly hierarchy because the user is involved in constructing the hierarchy when more than one consistent groupings are formed during the recognition phase. When a grouping is blamed, the source of the failure is in forming the grouping from a grouping schema in memory and the grouping is marked as negative application of the grouping schema.

7.2.3.1. Prevention of Precedence Schema Application

Equipped with the schemas learned from WEDGE assembly, the system is given a new target assembly to construct, BRACKET assembly. Fig. 7.11. shows the BRACKET assembly. The mnemonic names of parts are internally denoted by part numbers. Their correspondences are given in parentheses in the figure. Using GSCHEMA-1, a grouping, GI-2, is formed:

GI-2
INDEX: 2
GSCHEMAS: (GSCHEMA-1)
BASE: [PART4]
SATELLITES ([PART5] [PART6])
LIAISONS: (Liaison([PART4],[PART5]) Liaison([PART4],[PART6]))
PINSTS: (PI-2)

7.2.3.1.1. PSCHEMA-2 Update

An assembly sequence, PI-2, is formed by PSCHEMA-2 whose description of $diff-part2, [dloc-b($part3,$part2) < 0], is satisfied: [dloc-b([PART5],[PART6]) < 0]. That is, because [PART5] (bracket) is located closer to the base than [PART6] (nut), assemble the bracket before the nut. Fig. 7.12. shows the diagram representation of GI-1 and PI-1.

PI-2

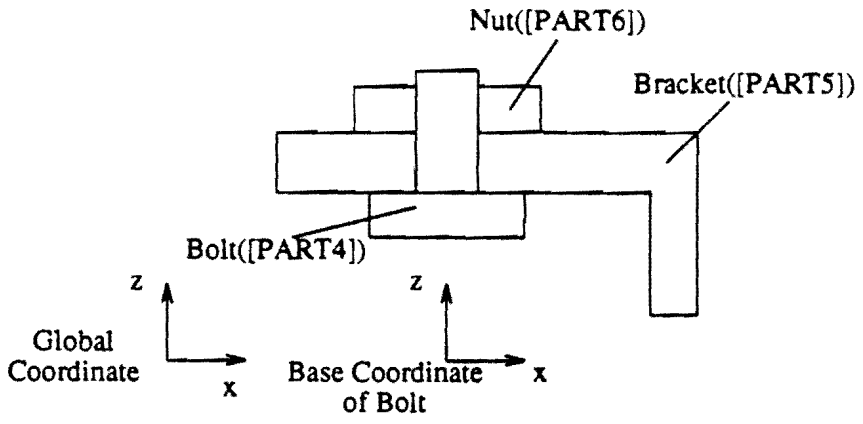


Figure 7.11. BRACKET Assembly

INDEX: 2
PSCHEMAS: (PSCHEMA-2)
GINST: GI-2
SEQUENCE: ([PART5] [PART6])
DIFF-SEQUENCE: (Diff-part([PART5],[PART6]))

PSCHEMA-2
INDEX: 2
GSCHEMA: GSCHEMA-1
SEQUENCE: ($part3 $part2)
DIFF-SEQUENCE: ($Diff-part2)
POS-PINSTS: (PI-2 PI-1)
NEG-PINSTS: ()

7.2.3.1.2. PSCHEMA-1 Update

Using PSCHEMA-1, a new assembly sequence, PI-3, is formed using its $Diff-part1, [dloc-g($part3,$part2) > 0].

PI-3
INDEX: 3
PSCHEMAS: (PSCHEMA-1)
GINST: GI-2

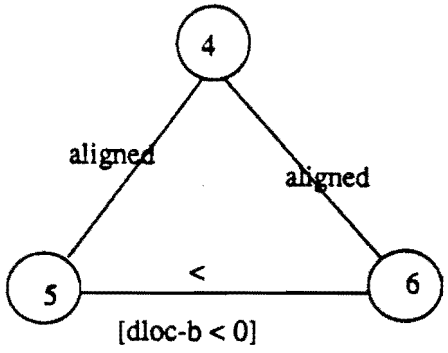


Figure 7.12. Assembly Episode from GSCHEMA-1 & PSCHEMA-2

SEQUENCE: ([PART6] [PART5])
DIFF-SEQUENCE: (Diff-part([PART6],[PART5]))

PSCHEMA-1
INDEX: 1
GSCHEMA: GSCHEMA-1
SEQUENCE: ($part3 $part2)
DIFF-SEQUENCE: ($Diff-part1)
POS-PINSTS: (PI-1)
NEG-PINSTS: (PI-3)

Using PI-3, the following assembly sequence is generated,

[[PART6],[PART4]] < [[PART5],[PART4]],

which fails and informed by the user who asserts the contradiction:

[[PART5],[PART4]] < [[PART6],[PART4]],

As a result, PI-3 is categorized as NEG-PINSTS of PSCHEMA-1.

7.2.3.1.3. After Updates

So far, PI-3 empirically invalidates the correctness of PSCHEMA-1 while PI-2 increases the confidence of the correctness of PSCHEMA-2. Using this increased knowledge, the system is given a

similar target assembly, SCREW assembly shown in Fig. 7.13. Using GSCHEMA-1 and PSCHEMA-2, the system correctly generates the following precedence relation:

[[PART45],[PART44]] < [[PART46],[PART44]]

7.2.3.2. Prevention of Grouping Schema Application

In the previous section, a violated precedence may depend on both a grouping and an assembly sequence segment. However, the sequence segment is blamed and updates. However, when all the assembly subsequences of a grouping are marked as negative application of their precedence schemas, the grouping itself is marked as negative application of its schema. Therefore, a grouping may be faulted directly from its participation in the assembly sequence by SPC and SIC or indirectly by propagating the faults of its assembly sequence segments when most of its assembly sequences are

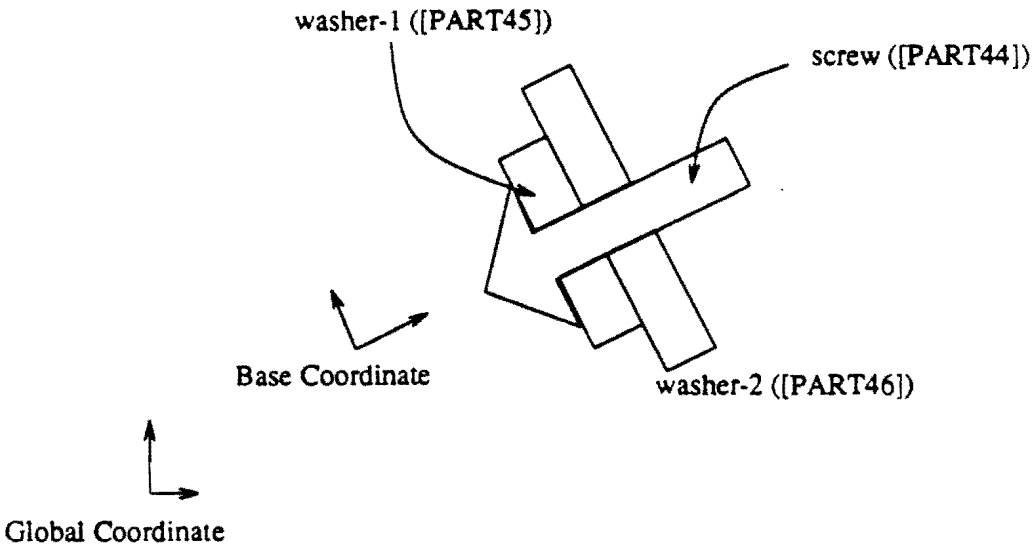


Figure 7.13. SCREW Assembly

empirically invalidated: a *secondary* credit-assignment. This section illustrates the failure of implicit grouping formation and the use of support structure in detail for the credit-assignment.

7.2.3.2.1. Credit Assignment for a Grouping

Consider an assembly example similar to Fig. 6.12 (a) in Fig. 7.14. Suppose only one grouping is used to create the hierarchy: GI-2:

GI-2

INDEX: 2
GSCHEMAS: (GSCHEMA-1)
BASE: [PART7]
SATELLITES: ([PART8] [PART9])
LIAISONS: (Liaison([PART7] [PART8]) Liaison([PART7] [PART9]))
PINSTS: (PI-2)

The hierarchy is formed using the implicit grouping assumption introduced in the previous chapter. The root is A3 and A2 is the subassembly.

A3

INDEX: 3
GINST: NIL

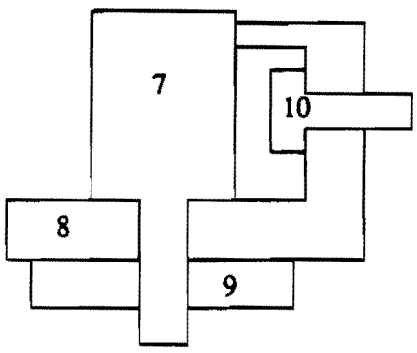


Figure 7.14. Target Assembly with Subassembly

BASE: [PART8]
SATELLITES: ([PART10])
SUBHIERARCHIES: (A2)

A2
INDEX: 2
GINST: GI-2
BASE: [PART7]
SATELLITES: ([PART8] [PART9])
SUBHIERARCHIES: ()

Using the hierarchy, the final assembly sequence is generated along with the support structure as shown in Fig. 7.15. Unfortunately, the assembly sequence fails and it is reported by the execution monitor as a precedence violation, those precedence relations supporting NOGOOD:

[[PART10],[PART8]] < [[PART8],[PART7]] and [[PART8],[PART7]] < [[PART10],[PART8]].

[[PART8],[PART7]] < [[PART10],[PART8]] of the final assembly sequence is responsible which is supported by SPC and GI-2. Therefore, GI-2 is ultimately responsible for the failure and thus, categorized as NEG-GINSTS of all the grouping schemas of GI-2, GSCHEMA-1.

GSCHEMA-1
```
INDEX:        1
BASE:         $part1
SATELLITES:     ($part2 $part3)
LIAISONS:       ($Liaison2 $Liaison1)
GINSTS:        ((GI-2
                 ($Liaison1 . Liaison([PART7] [PART9]))
                 ($Liaison2 . Liaison([PART7] [PART8]))
                 ($part3 . [PART8]) ($part2 . [PART9])
                 ($part1 . [PART7]))
                (GI-1
                 ($part1 . [PART1]) ($part3 . [PART2])
                 ($part2 . [PART3]) ($Liaison1 . Liaison([PART1] [PART3]))
                ($Liaison2 . Liaison([PART1] [PART2]))))
POS-GINSTS:     (GI-1)
NEG-GINSTS:     (GI-2) ;; GI-2 is negative instance of GSCHEMA-1
PSCHEMAS:       (PSCHEMA-2 PSCHEMA-1)
```

7.2.4. Construction of a New Macro-operator Schema

The above mechanism is mainly to create new *descriptions* of a schema while the structure of the schema remains untouched: it provides no mechanism to construct genuinely new structures for

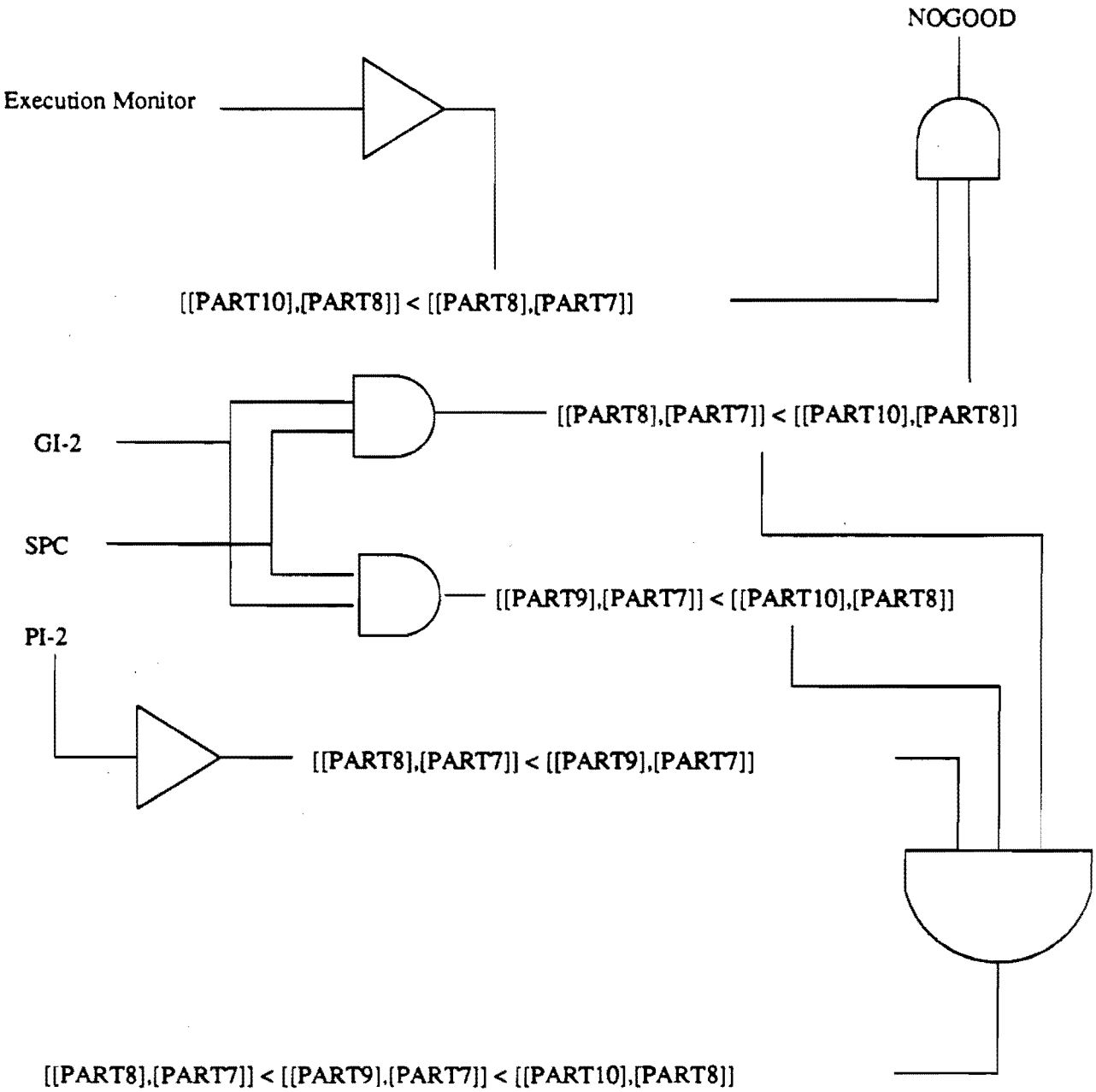


Figure 7.15. Support Structure for the Assembly Sequence

those groupings and precedence schemas in memory. In STRIPS, the search mechanism of the planner is the main construction mechanism for new macro-operators. This section introduces a new construction mechanism for constructing new macro-operators. The new macro-operators are used to learn new schemas, a form of *constructive induction.*

A *selective induction* is a form of induction that generates hypotheses containing only the descriptors present in the input data; a constructive induction is a form of induction that generates hypotheses including descriptors not present in the input data but constructed by the system [Michalski 1983]. So far, few effective constructive induction mechanisms were developed: e.g., constructing a new predicate for an extremal element of an ordered set of elements [Larson 1977]. Here, a new constructive induction mechanism is developed, based on the principle of the *selective merging*.

7.2.4.1. Selective Merging as a Constructive Induction Method

When multiple groupings and assembly subsequences are generated to build an assembly hierarchy, the system may notice that multiple groupings can be collapsed when their associated assembly subsequences can be merged into a continuous sequence, a selective merging based on *transitive* property of a temporal precedence. Note, this collapsed structure is a *deductive restructuring* of the instances generated from a planning scenario: the structures before the collapse logically implies the collapsed structure.

Furthermore, the new structure is *interesting* when it is not an instance of some existing schema because the collapsed structure create a longer and potentially more powerful macro-operator: the prediction of the macro-operator is more specific. Hence, the new instance should be an instance of some schema. As a result, a new schema, subsuming the interesting instance, is created.

This new schema is a refinement of the existing schemas whose instances are involved in the selective merging process. Therefore, the concept refinement is guided by the "interesting" instances

noticed during their usage. In general, the selective merging method can be applied to any structures containing predicates with a transitive property.

7.2.4.2. Example of Collapsing Structure during Assembly Planning

Consider that the planner was just trained with WEDGE assembly above: the memory contains GSCHEMA-1 and PSCHEMA-1. A new target assembly is given and shown in Fig. 7.16 (a). Its target assembly diagram is shown in Fig. 7.16. (b).

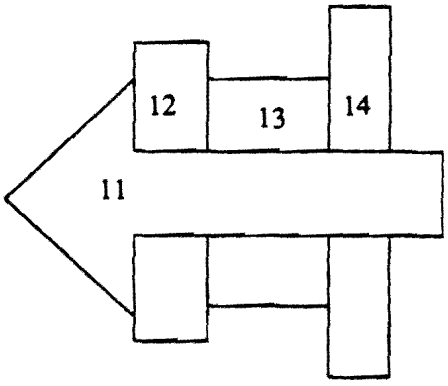
Using GSCHEMA-1, the following groupings, GI-2, GI-3, and GI-4 are formed:

GI-2
INDEX: 2
GSCHEMAS: (GSCHEMA-1)
BASE: [PART11]
SATELLITES: ([PART12] [PART13])
LIAISONS: (Liaison([PART11] [PART12]) Liaison([PART11] [PART13]))
PINSTS: (PI-2)

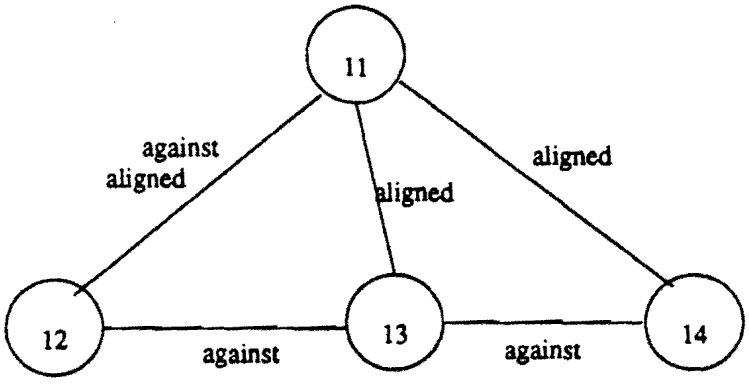
GI-3
INDEX: 3
GSCHEMAS: (GSCHEMA-1)
BASE: [PART11]
SATELLITES: ([PART12] [PART14])
LIAISONS: (Liaison([PART11] [PART12]) Liaison([PART11] [PART14]))
PINSTS: (PI-3)

GI-4
INDEX: 4
GSCHEMAS: (GSCHEMA-1)
BASE: [PART11]
SATELLITES: ([PART13] [PART14])
LIAISONS: (Liaison([PART11] [PART13]) Liaison([PART11] [PART14]))
PINSTS: (PI-4)

Their assembly sequences are generated by PSCHEMA-2; PSCHEMA-1 is not applicable because the locations of the parts with respect to the global coordinate are the same. These groupings and assembly sequences are shown in Fig. 7.17. Collapsing GI-2 and GI-3 into single structure would predict that [PART12] is the *first* satellite to be assembled from their PI-2 and PI-3. Collapsing GI-2 and



(a)



(b) Target Assembly Diagram

Figure 7.16 Arrow Assembly

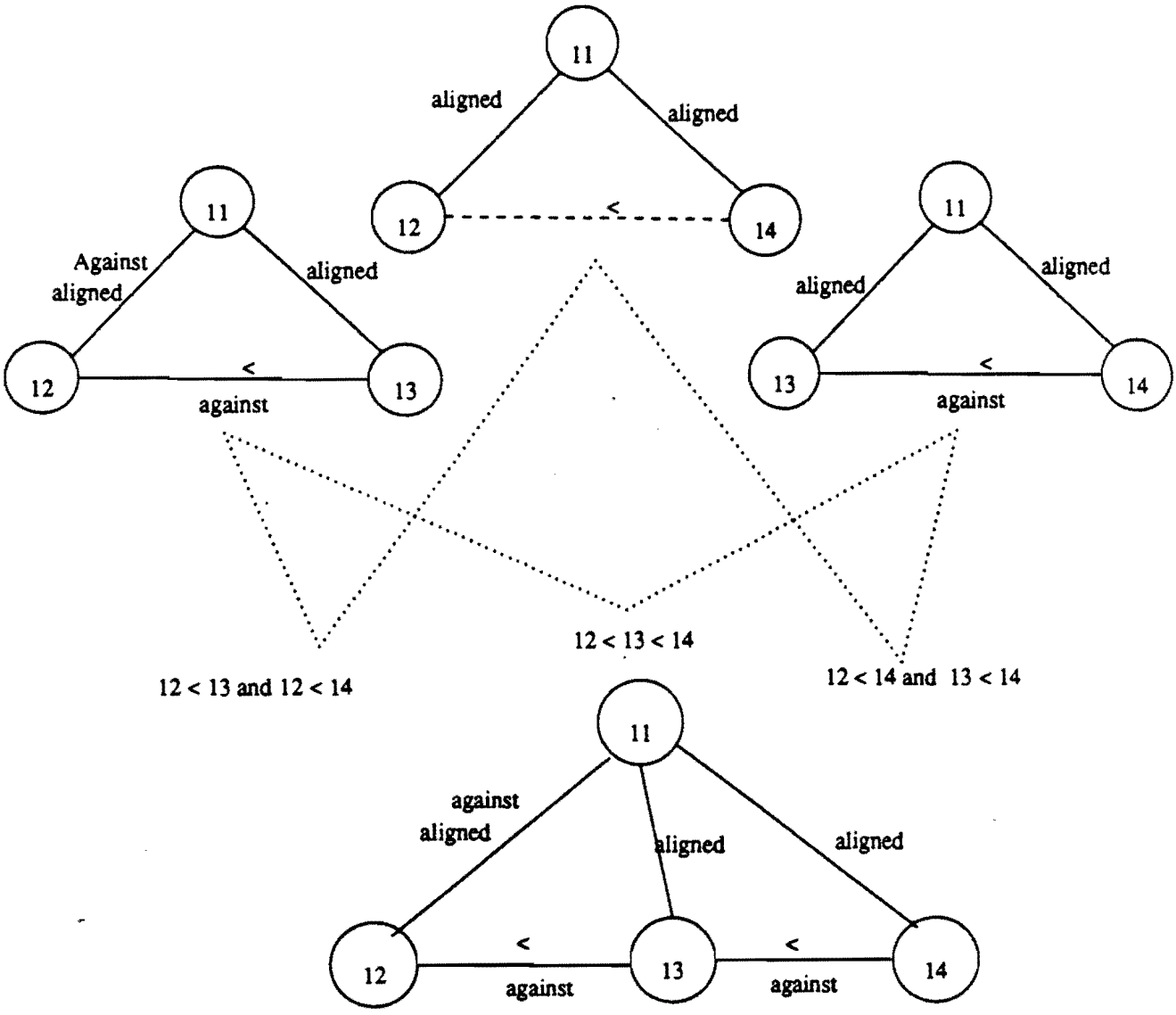


Figure 7.17 Collapsing Episodes using Transitivity

GI-4 would predict that [PART14] is the *last* satellite to be assembled from their PI-2 and PI-4. On the other hand, collapsing GI-2 and GI-4 would predict [PART12], [PART13], and [PART14] are assembled in sequence according to PI-2 and PI-4. NOMAD find collapsing GI-2 and GI-4 more

interesting than other collapses. Thus, creating GI-5 with the collapse:

GI-5
INDEX: 5
GSCHEMAS: (GSCHEMA-2) ;; *New schema is created*
BASE: [PART11]
SATELLITES: ([PART12] [PART13] [PART14]) ;; *Three satellites*
LIAISONS: (Liaison([PART11] [PART12]) Liaison([PART11] [PART13]) Liaison([PART11] [PART14]))
PINSTS: (PI-5)

From the new grouping (GI-5) and its assembly sequence (PI-5), the system forms a novel grouping schema, GSCHEMA-2:

GSCHEMA-2
INDEX: 2
BASE: $part4
SATELLITES: ($part5 $part6 $part7)
LIAISONS: ($Liaison5 $Liaison4 $Liaison3)
GINSTS: ((GI-5 ($part4 . [PART11]) ($part7 . [PART14])
($part6 . [PART13]) ($part5 . [PART12])
($Liaison3 . Liaison([PART11] [PART12]))
($Liaison4 . Liaison([PART11] [PART13]))
($Liaison5 . Liaison([PART11] [PART14]))))
POS-GINSTS: (GI-5)
NEG-GINSTS: NIL
PSCHEMAS: (PSCHEMA-4 PSCHEMA-3)

GSCHEMA-2 is a *specialization* of GSCHEMA-1; however, the refinement is guided by the interestingness of information content of the predictability: the more specific the predicted information, the more interesting it is.

CHAPTER 8.

SUMMARY AND FUTURE WORK

To be autonomous, AI systems should exhibit both planning and learning capabilities. Planning is necessary because it is impossible to program every conceivable course of actions to be taken by the system and planning mechanism enables to the system to "program" itself a new course actions for new task situations. Furthermore, the problem solving mechanism used by a planner is inadequate for the simple planning tasks people encounter everyday. It is shown that such a mechanism would be combinatorially explosive even with a simplifications [Chapman, 1985]. However, we are existence proofs that there exist such a mechanism and hence, it can be automated..

I believe that a planning mechanism includes an amalgamation of diverse mechanisms including a problem-solving mechanism. In fact, if a plan requires a lot of problem-solving, it is unlikely to succeed because we are bound to make a mistake. For most planning tasks, people may use of "canned" plans in memory. We only resort to a problem-solving activity when we can not proceed with such a "lazy" attitude. Therefore, a large portion of our planning activities involves recognizing similar previous planning episodes and applying them. This requires a learning mechanism that can store and recall episodes for similar planning situations.

The planning situation may involve a goal description and a sequence of actions taken to achieve the goals. The sequence of actions is a structural description with "next-to" relations [Michalski & Ko & Chen, 1987]. For the assembly planning, the goal description is a structural description as well: the geometric, topologic, and kinematic description of a target assembly.

In general, the learning mechanism for a planner should be capable of learning structural descriptions. Learning structural concepts has been a long standing research problem in machine

learning. Previously, learning from spatially structured objects and learning from temporally structured objects have been studied independently. Initially, the former concentrated on domain-independent methods for learning structural concepts [Winston, 1970; Larson, 1977; Stepp, 1984]. Recently, the latter concentrated on domain-independent methods for learning from a temporal sequence of objects, qualitative process prediction [Michalski & Ko & Chen, 1987; Chen, 1988]. This research focused on learning the structural descriptions involving both the spatial and temporal relationships as planning scenarios.

Furthermore, most learning systems are designed primarily for answering **how** to learn a concept with different algorithms where the user determines for the system **when** to learn and **what** concept to learn. This paradigm by itself is inadequate for a planner that requires many different concepts in different contexts: it requires "multi-concept" learning from a single observation by a **constructive learning** mechanism like NOMAD.

8.1. Summary of NOMAD

This section gives intuitions behind the methodological paths taken by NOMAD.

8.1.1. Model of Problem-Solving

When encountered with a new problem in the environment, a human as an intelligent being attacks the problem in a systematic way. First, he tries to break the problem into subproblems for which he has solution methods in memory. Second, for those subproblems for which he does not recognize any stored solutions, he makes up a new solution method using some general problem-solving techniques such as weak methods or asks the teacher for a solution. Finally, all the methods used to solve subproblems of the overall problem are combined into a coherent procedure that solves the whole problem.

The problem-solver may experience a success or a failure of this procedure. This experience is brought to bear in the future problem-solving tasks by assimilating its experience: the assimilation process involves restructuring the memory structure, containing subproblems and their solution methods, to reflect the new experience. This memory structure is, then, used to solve new problems as discussed above. As a result, the performance of the problem-solver depends on the extent to which the memory structure can be used to decompose the problem into proper subproblems and suggest their correct solutions.

Then, the learning task is to construct a memory structure, encompassing all the problem-solving situations that are going to be encountered by the problem solver, that can be used to pose correct solutions. Empirical planning method was based on this model of problem-solving where the task schema corresponds to a subproblem and its plan schema corresponds to the correct solutions for the subproblem.

8.1.2. Positivism

NOMAD learns by noticing failures to execute the planned action and then detecting the cause of the failure to facilitate learning. Recovery is not stressed after failure but rather the failure is avoided by constraining the schema applicability conditions. Here, executing an action is an experiment to collect information and to modify planning strategies to avoid the failure in a similar situation. The use of positive and negative planning episodes by the empirical learning method is explained in this section.

8.1.3. Necessity vs Sufficiency: Success vs Failure

There may be many different sufficient explanations why an object is falling to the ground during a mating operation: the object is thrown into the ground, dropped, a supporting object is removed,

the earth pulls the object, objects with mass attracts each other and as many as your imagination allows. The point is that the "right" level of explanation will be a function of our knowledge of the physical world. That is, any one of the theories behind the explanation is sufficient as far as the task using the theory is restricted enough.

On the other hand, the number of possible reasons that the object may not hit the ground is much larger. Therefore, it would be inappropriate to search with a weak method, basic mechanisms of most empirical inductive learning systems. Hence, the empirical assembly planner concentrated on learning a *sufficient* condition of a schema from *successful* application of the schema and a *necessary* condition of the schema from *failed* applications.

8.1.3.1. Empiricism for Success and Analysis for Failure

Those schemas in planner's memory as the task theory predict positive outcomes rather than negative ones. Alternatively, the failure itself may be compiled into schemas that are used to prune faulty plans. Intuitively, the space of the possible reasons for the failures in the assembly task domain is immense, especially without a closed-world assumption. However, there are relatively few sufficient reasons for the failure. For example, consider all the possible reasons that a mating operation for an object may fail. Hence, learning useful failure rules requires more guidance from the background knowledge in order to control the generalization process like an explanation-based learning (EBL) approach [Mitchell & Keller & Kedar-Cabelli 1986; Dejong & Mooney 1986] rather than an empirical generalization approach.

On the other hand, both the space of the possible reasons for the success and the number of sufficient reasons of the success are large as the range of context is restricted. For example, if the type of material is restricted to iron, the sizes of components are sufficient to predict their weights.

Because the description space for explaining a failure is large and only a small area of the space represents a causal explanation of the failure, the target concept, applying an empirical generalization method, based on a search mechanism, is unlikely to find the target concept. In contrast, the failure should be carefully analyzed to uncover the proper element to generalize: maybe using an EBL method. Otherwise, arbitrary generalization would likely lead to epiphenomenal explanations for the failure.

The analysis by an EBL method requires complete and sound diagnostic rules before learning the failure concept: namely the collision physics. But it is determined in the previous section that reasoning from the collision physics is intractable. Here, compiling the failure itself is postponed until such diagnostic rules are developed. In short, the planning experiences are empirically generalized to form a new schema that is used to generate a successful plan rather than to prune a faulty plan.

8.2. Future Works

The current research work suffers many limitations. The following sections are intended to point out the limitations and their extensions of the current work.

8.2.1. Trajectory Planning

The single most glaring omission of NOMAD is that the trajectory planning is simulated by the user. This turned out to be a significant limitation that affected the entire system. The learner was affected due to the lack of the causal theory of the environment (to be discussed in the next section). The planner was affected because the execution had to be simulated with a user input of precedence violations. Although the trajectory planning should report an execution failure as a precedence violation anyway, it would have been less cumbersome to conduct experiments if an automatic trajectory planner was at hand.

With a trajectory planner, the main difficulty is that the system has to infer which precedences that are responsible for assembling the obstructing components. An uncontrolled spatial search for the collision free path would post possibly all the outer layer components as candidate components responsible for the failure. Furthermore, removal of anyone of those components would not guarantee a collision free path, violating the credit-assignment assumption. A possible solution is to associate typical approach paths with each mating pairs so that the trajectory planner use the paths when available. All the components in any of the paths from some approach point outside to the assembled position inside are credited for the failure and hence, all the precedence relations used to place the culprit components are responsible for the failure. In short, an immediate research task is to incorporate a trajectory planner.

8.2.1.1. Causal Theory of the Environment

The attributes defined for the planning episodes in the previous chapter were defined cleverly by the user equipped with the knowledge of the nature of the assembly task at hand so that the desired final outcome was generated. Hence, the user is partially responsible for the successful learning scenario. The learning mechanisms are also partially responsible for the final outcome.

However, the implicit knowledge used by the user to generate plausible attributes should be explicitly encoded into the system as a causal theory of the environment. This causal theory allows the system to reason from the nature of the failure or success of the assembly planning episode to postulate a "good" way to assimilate the planning episode, a mental representation of its experience. The theory may require the entire naive physics [Hayes, 1979].

In the context of the assembly planning where the environment is governed by the collision physics, the causal theory amounts to axiomatizing the trajectory planning process. This effort is necessary to achieve the kind of competence humans exhibit in the assembly tasks.

8.2.2. Hybrid-Approach to Assembly Planning

For assembly planning, two complimentary approaches were studied: one was based on detecting the precedence constraints from the spatial interference checking and the other was based on posting planning episodes (an assembly sequence) that were recognized using the empirically learned planning scenarios in memory as a candidate plan structure. The former approach is accurate but intractable. The latter approach is less accurate but for those assembly problems it solves, it is efficient.

Currently, two approaches are separate. However, they operate under a similar representational framework, the target assembly diagram. As a result, it is foreseeable that these two approaches can be profitably combined into a hybrid approach where the planner checks the spatial interferences between assembly steps for those parts of the target assembly without appropriate planning scenarios in memory but use the scenarios where appropriate.

8.2.3. Automatic Construction of Abstraction Hierarchies

An assembly hierarchy is an abstraction hierarchy. Although this research only constructs the hierarchy semi-automatically, it is feasible to foresee an automatic construction of an assembly hierarchy. Then, it may shed light to the levels of abstraction issues for problem-solving and representational issues for learning.

Subassembly precedence and independence constraints are specialization of the more powerful abstraction principles. Furthermore, their validation is monitored by the violation of SPC or SIC. As a result, the abstraction formation process is criticized and thereby, can be improved with experience. Furthermore, it would be interesting to apply SPC and SIC to domains other than assembly planning domain.

8.2.4. Intelligent Computer-Aided Design

Eventually, this research may contribute towards bringing an *integrated design and manufacturing environment* to the desk-tops of the designers of mechanical assemblies. The knowledge about building an assembly structure may provide immediate feedback of the *manufacturability* of a design without having to consult a production engineer. On the other hand, an automated assembly manufacturing device may benefit "design for assembly" principle where design techniques are applied in order to facilitate automated assembly tasks.

8.2.5. Functional Assembly Design

The functional and behavioral knowledge during the assembly design should be brought to bear in determining subassemblies. The kinematic behavior of individual components are designed to serve as a part of some kinematic behavior and a subassembly as a whole may serve a functional role. An automobile engine may transmit up-and-down motion to cylindrical motion while its functional role is to the car. An encapsulation of the mechanical behavior, modularity, may determine subassemblies. Other considerations for subassemblies are replaceability and serviceability.

As a start, the assembly design proceeds by collapsing modularized design of subassemblies via the function sharing [Ulrich & Seering, 1988] mechanism. However, the collapsing should be allowed only when the resulting structure does not prohibit assemblability. The grouping and its associated assembly sequences may represent one of the repertoire of subassemblies with the assembly procedures: "canned" designs. Then, collapsing among these designs would be tested whether the merged assembly procedure is viable (no violation of SIC or SPC).

8.2.6. Mechanism Simulation

This mechanical behavior causes alternative configurations of the mechanical device and accessibility may change. For example, the release mechanism of a seat belt cause two different states of the seat belt configuration: one is to release the belt and the other is to prevent the disassembly of the belt.

8.2.7. Extension of the Assembly Criterion

Currently, the sole assembly criterion is the accessibility. It is necessary to extend the criteria with dynamic notions such as stability and friction. For the time being, the notion of rigid body assumption should stay until other assembly criteria are better understood.

8.2.8. Case-based Reasoning

The empirical generalization is a risky business, especially with many possible generalizations. The current learning mechanism is too "eager" to generalize. It should be moderated by storing the instances until the generalization is justified.

8.2.9. Robot Programming

If NOMAD be extended to a real robot, each assembly task should be converted to the robotic instructions in some robot language that may be executed to actuate the robot. Therefore, the amount of conversion depends on the level of instructions that are understood by the robot system. Many robot languages are currently available or proposed with varying degrees of sophistication.

Various levels of robot programming can be viewed hierarchically so that the higher level language embeds the lower level robot language. Then, the high level robot language system is

responsible for converting the high level instructions into the joint level control of the robot manipulator. A high level task such as an assembly task is converted to robotic instructions, done either by the task planner or the robot programming system. It is not clear to what extent of this conversion is carried out by a programming system but the separation between the joint level and manipulator level robot programming is well-understood. Here, various levels of robot programming systems are discussed from low to high levels.

Joint level programming is where the motions of the robot are specified in terms of the required angular positions of its various joints. This level of interface to the robot system is programmed entirely by teaching, a teach pendant method. The robot programming is hard to incorporate sensory feedback to control the robot. Many paint spraying robots are of this type.

Manipulator level programming is where the motion of the robot is specified in terms of the required position of the end-effector of the robot, expressed in some coordinate system. This involves inverse kinematics from the coordinate location to the joint angles which may require expensive computations in general. Recently, this difficulty is alleviated by a clever design of the manipulator that decouples translational and rotational portion of the computation, a scara (selective compliant articulated robot for assembly) robot. This is the general level of the interface to most commercially available robot today.

Object level programming is where the instructions are given in terms of the spatial relations that are specified between features of the objects to be assembled. This is called task level programming [Lozano-Perez 198] but the term is overly general for its intended meaning. Here, we use object level programming (OLP) [Smither & Malcolm 1987] because the term gives more information about its intended meaning: the level of instruction is object oriented rather than the robot oriented.

OLP requires much more sophisticated mechanisms than manipulator level programming systems. First of all, an inference engine capable of reasoning over spatial relationships of objects to

deduce their positions is required. Then, a trajectory planner that finds a collision free trajectory is required. In addition to a collision free trajectory planner, a grasp planner and a compliant motion planner are required as well. The coupling between object and manipulator levels is incomplete and remains as an open research problem both in robotics and AI.

Assembly level programming is where an instruction is simply the specification of the finally assembled state. This requires assembly planning. Note that the three previous levels have been simple escalations in the sophistication of defining the motions of the robot in space, whereas assembly level is also concerned with planning the assembly sequence. In general, as the robot programming becomes more sophisticated, it encompasses more of the planning component of the task. NOMAD, when fully developed and interfaced with an actual robot, can be viewed as an assembly level programming system.

APPENDIX A.

USER'S GUIDE

NOrmative Mechanical Assembly Device (NOMAD) is the name of the prgram. It is written in pure common lisp with a few exception of window operations for graphics package. The program runs in Symbolics 3640 with version 6.1 operating system.

A.1. Running GMS

The geometric modeling system is loaded by:

1. Load file symb:>ko>gms>wing1.bin
2. Load file symb:>ko>imsa>test-gms.lisp
3. (new-window)

Now, you can use a mouse to create a window that displays scenes on the screen. Then:

4. (clear) ;; clears the screen

In test-gms.lisp, several objects are modeled and one of them is called neun:

5. (neun)
*6. (dbody *obody*) ;; display *obody**
*7. (dbody *pbody*) ;; display *pbody**

Steps 6 and 7 will show the two objects about to participate in boolean operation of the GMS: minus operator (this is called as difference operator in the thesis).

*9. (setq *rbody* (minus *obody* *pbody*))*
*10. (dbody *rbody*)*

Step 9 assigns the resulting body to *rbody* after boolean operation and step 10 displays the object.

A.2. Running Spatial Reasoning Engine

The location map and reasoning about mating condition can be run by:

1. Load file symb:>ko>gms>wing1.bin
2. Load file symb:>ko>imsa>graph.bin

The raw screen outputs from the system capturing the bench-beam example in Chapter 4 are provided in Appendix B.

A.3. Running Planning and Learning Scenarios

All the examples in Chapter 7 can be run by:

1. Load file symb:>ko>imsa>atms.bin
2. Load file symb:>ko>imsa>atre-var.bin
3. Load file symb:>ko>imsa>nomad.bin
4. Load file symb:>ko>imsa>nomad-decl.lisp
5. Load file symb:>ko>imsa>test-nomad.lisp

The trace of the session is provided in Appendix C.

APPENDIX B.

Screen Outputs for Spatial Reasoning

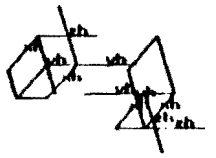
The following four pages show a raw interaction with the system solving the bench-beam example discussed in the spatial reasoning engine chapter.

The first page shows nodes 3, 4, 12, and 15 in window 3 using "display-node" routine. Their positions with respect to *world* is represented by homogeneous transformation matrices that are computed by a routine "frame" that computes relative position between any two nodes in the position map. The matrices are displayed by "matrix-printer"; the matrix is repersented in a row major convention instead of the column major convention followed in the thesis.

The second page starts with "(c)" that clears the window 3. The routine, "assert-against-into-joint", asserts an against relation between faces. *mating* contains all the mating structures specified. Against-1 is created for the against mating condition specified between nodes 3 and 12. The structure contains rpos slot that contains a homogeneous transformation matrix for the mating condition. The third page specifies an against mating condition between nodes 4 and 15 and againt-2 is created as a result. The final page shows the joint (called a kinematic link in the thesis) relation between the base nodes: 2 and 11 as a result of combining the againt-1 and againt-2.

```
(display-node (node 3))
313
172
(display-node (node 4))
313
172
(display-node (node 12))
380
241
(display-node (node 15))
396
245
(matrix-pprinter (frame (node 3) *world*))
       col-0  col-1  col-2  col-3
row-0    1      0      0      0
row-1    0      0     -1      0
row-2    0      1      0      0
row-3    0      0.5    1      1
NIL
(matrix-pprinter (frame (node 4) *world*))
       col-0  col-1  col-2  col-3
row-0    1      0      0      0
row-1    0      1      0      0
row-2    0      0      1      0
row-3    0      1      0.5    1
NIL
(matrix-pprinter (frame (node 12) *world*))
       col-0  col-1  col-2  col-3
row-0    1      0      0      0
row-1    0     -1      0      0
row-2    0      0     -1      0
row-3    0      2      0      1
NIL
(matrix-pprinter (frame (node 15) *world*))
       col-0  col-1  col-2  col-3
row-0   -1      0      0      0
row-1    0      0      1      0
row-2    0      1      0      0
row-3    1      2.5    0      1
NIL
```

Common Lisp Listener 1



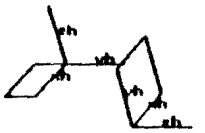
Window 3

```
  77 blocks in 5 files.

(display-node (node 15))
(Show Directory (files [default SYMB:>ko>*.*.*]) a*

SYMB:>ko>a*.*.*
  2047 free, 9643/12490 used (77%)

      a1.press.1       25 100073(0)      1   00/00/87 21:50:35 (00/0
0/87)
      a2.press.1       25 100073(0)      1   00/00/87 21:53:26 (00/0
0/87)
      a3.press.1       25 100073(0)      1   00/00/87 21:54:53 (00/0
0/87)
      algebra.directory.1     1   DIRECTORY 1   04/14/87 00:55:01 X
=07/31/87
      assembly.directory.1    1   DIRECTORY 1   04/23/86 14:50:27
X=07/31/87
```

Common Lisp Listener 2

```
* (c)
NIL
* (display-node (node 4))
313
172
* (display-node (node 15))
396
245
* (assert-against-into-joint (node 15) (node 4))
1.0000000000000027d0
* *mating*
(AGAINST-2(<Node 15>,<Node 4>) AGAINST-1(<Node 12>,<Node 3>))
* (describe (car *mating*))
AGAINST-2(<Node 15>,<Node 4>) is a MATING
   INDEX:                          2
   PART1:                          <Node 15>
   PART2:                          <Node 4>
   BUF-VARS:                       (0:|theta1501| 0:|tr-x1502| 0:
|tr-y1503|)
   RANGES:                         ((MAX-X . 3.0) (MIN-X . -1.0)
(MAX-Y . 0.0d0) (MIN-Y . -3.5d0) (MAX-Z . 1.0000000000000027d0)
 (MIN-Z . 1.0000000000000027d0))
   TYPE:                           AGAINST
   APOS:                           8<ART-0-4-4 17541214>
AGAINST-2(<Node 15>,<Node 4>) is implemented as an ART-0 type a
rray.
It uses ZARRAY-BIGFATCH-WORD; it is 0 elements long.
AGAINST-2(<Node 15>,<Node 4>)
* (matrix-pprinter (mating-rpos (car *mating*)))
       col-0  col-1  col-2  col-3
row-0    (COS theta1501)      (- (SIN theta1501))  0      0
row-1    0      0      -1     0
row-2    (SIN theta1501)      (COS theta1501)      0      0
row-3    (- 1.0 (+ (COS theta1501) tr-x1502))      (+ -2.0d0 (+
 (SIN theta1501) tr-y1503))   1      1
NIL
*
```

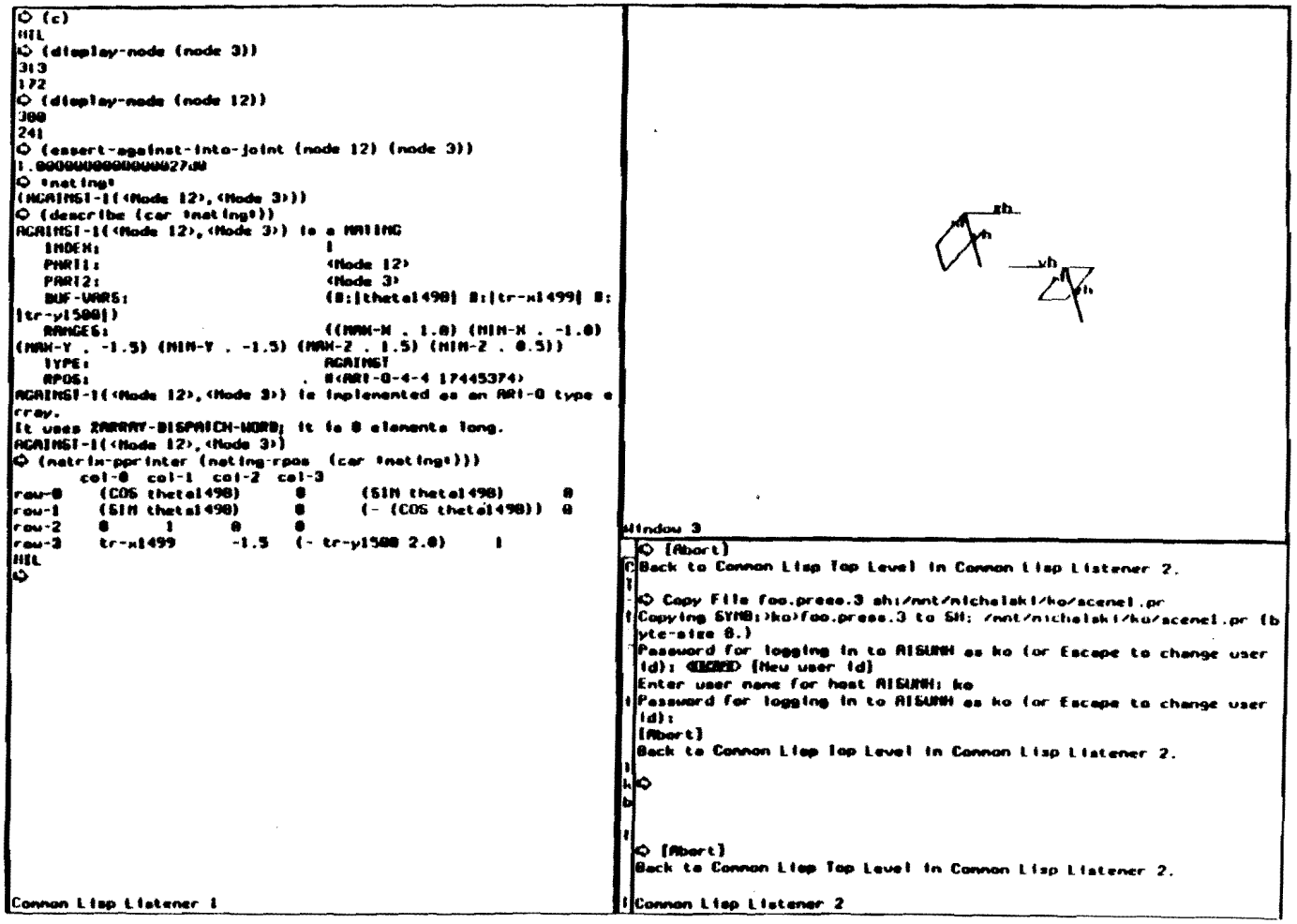
Common Lisp Listener 1



Window 3

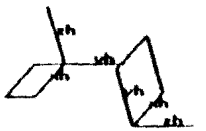
```
0/07)
  25 blocks in 1 file.

*
Password for logging in to AISUNH as ko (or Escape to change user
id): [User] [New user id]
Enter user name for host AISUNH: ko
Password for logging in to AISUNH as ko (or Escape to change user
id):
[Abort]
Back to Common Lisp top Level in Common Lisp Listener 2.

* Show Directory (files [default SYMB:>ko>*.*.*]) al*

SYMB:>ko>al*.*.*
  2897 free, 9593/12490 used (76%)

      al.press.1      25 100073(0)     !   00/00/07 21:50:35 (00/0
```

Common Lisp Listener 2


(c)
HIL
(display-node (node 3))
313
172
(display-node (node 12))
300
241
(assert-against-into-joint (node 12) (node 3))
1.0000000000000027dN
mating
(AGAINST-1(<Node 12>,<Node 3>))
(describe (car *mating*))
AGAINST-1(<Node 12>,<Node 3>) is a MATING
INDEX: 1
PART1: <Node 12>
PART2: <Node 3>
BUF-VARS: (B:|theta1498| B:|tr-x1499| B:|tr-y1500|)
RANGES: ((MAX-X . 1.0) (MIN-X . -1.0) (MAX-Y . -1.5) (MIN-Y . -1.5) (MAX-Z . 1.5) (MIN-Z . 0.5))
TYPE: AGAINST
RPOS: . #<ART-Q-4-4 17445374>
AGAINST-1(<Node 12>,<Node 3>) is implemented as an ART-Q type array.
It uses ZARRAY-DISPATCH-WORD; it is 8 elements long.
AGAINST-1(<Node 12>,<Node 3>)
(matrix-pprinter (mating-rpos (car *mating*)))
col-0 col-1 col-2 col-3
row-0 (COS theta1498) 0 (SIN theta1498) 0
row-1 (SIN theta1498) 0 (- (COS theta1498)) 0
row-2 0 1 0 0
row-3 tr-x1499 -1.5 (- tr-y1500 2.0) 1
HIL
Common Lisp Listener 1
xh
yh
yh
xh
Window 3
[Abort]
Back to Common Lisp Top Level in Common Lisp Listener 2.
Copy File foo.press.3 sh:/mnt/nichalak1/ko/acenel.pr
Copying SYMB:>ko>foo.press.3 to SH: /mnt/nichalak1/ko/acenel.pr (byte-size 8.)
Password for logging in to AISUNH as ko (or Escape to change user id): [ESCAPE] [New user id]
Enter user name for host AISUNH: ko
Password for logging in to AISUNH as ko (or Escape to change user id):
[Abort]
Back to Common Lisp Top Level in Common Lisp Listener 2.
[Abort]
Back to Common Lisp Top Level in Common Lisp Listener 2.
Common Lisp Listener 2

```
> (describe (car *joints*))
JOINT1(<Node 11>, <Node 2>) is a JOINT
   INDEX:                          1
   TYPE:                           NIL
   PART1:                          <Node 11>
   PART2:                          <Node 2>
   MATINGS:                        (AGAINST-2(<Node 15>,<Node 4>)
AGAINST-1(<Node 12>,<Node 3>))
   INV-VARS:                       NIL
   INF:                            #<ART-Q-4-4 1754174?>
   RANGES:                         ((MAX-X . 1.0) (MIN-X . -1.0)
(MAX-Y . -1.5) (MIN-Y . -1.5) (MAX-Z . 1.0000000000000027d0) (M
IN-Z . 1.0000000000000027d0))
JOINT1(<Node 11>, <Node 2>) is implemented as an ART-Q type arr
ay.
It uses 2ARRAY-BIGFATCH-WORDs; it is 9 elements long.
JOINT1(<Node 11>, <Node 2>)
> (matrix-pprinter (joint-dof (car *joints*)))
        col-0  col-1  col-2  col-3
row-0     1      0      0      0
row-1     0      0     -1      0
row-2     0      1      0      0
row-3    tr-#1499      -1.5   1      1
NIL
>
```

Common Lisp Listener 1



Window 3

```
0/07)
  25 blocks in 1 file.

>
Password for logging in to AISUNH as ko (or Escape to change user
id): [redacted] (New user id)
Enter user name for host AISUNH: ko
Password for logging in to AISUNH as ko (or Escape to change user
id):
[Abort]
Back to Common Lisp Top Level in Common Lisp Listener 2.

> Show Directory (files [default SYMB:>ko>*.*.*]) al*

SYMB:>ko>al*.*.*
  2097 free, 9593/12490 used (76%)

      al.press.1      25 100073(0)      1   00/00/07 21:50:35 (00/0

Common Lisp Listener 2
```

APPENDIX C.

TRANSCRIPT OF INTERACTION WITH NOMAD

This appendix illustrates a raw transcript with NOMAD during learning and planning stage except comments that are delineated by semi-colons.

C.1. Learning Precedence Schema

This section shows NOMAD's capability to learn a generalized knowledge and store it into its memory and uses it to solve new problems.

```
(bb) ;; Initialize the memory: Big Bang
NIL
(nomad-decl) ;;Declare Attributes
<NOM (LIAISON)>
(bell-ex1)
BELL1-ASSEMBLY
(describe bell1)
BELL1-ASSEMBLY is a ASSEMBLY
  NAME:          BELL1
  PARTS:         ([PART19] [PART20] [PART21])
  LIAISONS:        (Liaison([PART20] [PART21])
Liaison([PART19] [PART21]) Liaison([PART19] [PART20]))
  HIERARCHIES:     NIL ;; No assembly hierarchy yet
BELL1-ASSEMBLY is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 5 elements long.
BELL1-ASSEMBLY

(apg bell1) ;; Assembly Procedure Generation of bell1
>>Please choose one component for the base component
 Please choose the corresponding number:
 1: [PART19]
 2: [PART20]
 3: [PART21]
Enter the number of your choice (0 if nothing) > 1

>>Please choose satellite components
Please choose among the following objects by specifying their
corresponding numbers:
 1: [PART20]
```

```
2: [PART21]
0 if finished
 Choose a number> 1
 Choose a number> 2
Do you want to stop? (Y or N) Yes.
 Please choose in sequence
Please choose among the following objects by specifying their
corresponding numbers:
 1: [PART21]
 2: [PART20]
 0 if finished
 Choose a number> 2
 Choose a number> 1
(PINST PI-1)
(GINST GI-1)
[[PART20],[PART19]] < [[PART21],[PART19]]
EXECUTION-MONITOR
 Please input execution failure assertions justified by *execution-monitor*
 > bye ;; The assembly procedure is successfully executed.
NIL
;; The bell1 assembly structure after the planning process is completed
(describe bell1)
BELL1-ASSEMBLY is a ASSEMBLY
  NAME:          BELL1
  PARTS:         ([PART19] [PART20] [PART21])
  LIAISONS:        (Liaison([PART20] [PART21])
Liaison([PART19] [PART21]) Liaison([PART19] [PART20]))
;; A new assembly hierarchy, A1, is created after the planning process
  HIERARCHIES:    (A1)
BELL1-ASSEMBLY is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 5 elements long.
BELL1-ASSEMBLY
;; Describe A1
(describe (car (assembly-hierarchies bell1)))
A1 is a HIERARCHY
  INDEX:         1
  GINST:         GI-1
  BASE:          [PART19]
  SATELLITES:    ([PART21] [PART20])
  AUXLIAISONS:    NIL
  SUBHIERARCHIES: NIL
A1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.
A1
(d *$gschema*) ;; Display all the known gschemas in memory
GSCHEMA-1 is a $GSCHEMA
  INDEX:         1
  BASE:          $part1
  SATELLITES:    ($part2 $part3)
  LIAISONS:      ($Liaison2 $Liaison1)
```

```
  GINSTS:        ((GI-1 ($part1 . [PART19]) ($part3 .
[PART20]) ($part2 . [PART21]) ($Liaison1 . Liaison([PART19]
[PART21])) ($Liaison2 . Liaison([PART19] [PART20]))))
;; The grouping instance formed from applying gschema1 was successful
  POS-GINSTS:     (GI-1)
;; No negative evidence for gschema1 has been accumulated yet
  NEG-GINSTS:     NIL
;; Two possible generalized precedence schemas are created
;; for this gschema
  PSCHEMAS:       (PSCHEMA-2 PSCHEMA-1)
  SUPERS:         NIL
  SUBS:           NIL
GSCHEMA-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 11 elements long.
NIL
(d *$pschema*) ;; Display all the known pschemas in memory
PSCHEMA-2 is a $PSCHEMA
  INDEX:          2
  GSCHEMA:        GSCHEMA-1
  SEQUENCE:       ($part3 $part2)
  DIFF-SEQUENCE:  ($Diff-part2)
  PINSTS:         (PI-1)
  POS-PINSTS:     (PI-1)
  NEG-PINSTS:     NIL
  SUPERS:         NIL
  SUBS:           NIL
PSCHEMA-2 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 10 elements long.

PSCHEMA-1 is a $PSCHEMA
  INDEX:          1
  GSCHEMA:        GSCHEMA-1
  SEQUENCE:       ($part3 $part2)
  DIFF-SEQUENCE:  ($Diff-part1)
  PINSTS:         (PI-1)
  POS-PINSTS:     (PI-1)
  NEG-PINSTS:     NIL
  SUPERS:         NIL
  SUBS:           NIL
PSCHEMA-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 10 elements long.
NIL
(bracket-ex) ;; Pose a similar assembly task, bracket assembly
BRACKET-ASSEMBLY
(apg bracket) ;; Generate assembly procedure for the bracket
(PINST PI-2)
(GINST GI-2)
[[PART5],[PART4]] < [[PART6],[PART4]]
EXECUTION-MONITOR
· Please input execution failure assertions justified by *execution-monitor*
```

```
 > bye ;; The assembly procedure succeeds
(PINST PI-3)
(GINST GI-2)
[[PART6],[PART4]] < [[PART5],[PART4]]
EXECUTION-MONITOR
 Please input execution failure assertions justified by *execution-monitor*
 > (Failed-with ([[PART6],[PART4]] < [[PART5],[PART4]]))
 > bye
%Got nogoods:(E-6)  ;; The assembly proceudre fails
NIL
(print-env *before-nogoods*) ;; List possible blames
E-6:* {EXECUTION-MONITOR, (GINST GI-2), (PINST PI-3)}
NIL
(d *$pschema*)
;; New contents of pschemas showing that pschema2 has no
;; incorrect application while pschema-1 contains incorrect
;; application
PSCHEMA-2 is a $PSCHEMA
  INDEX:          2
  GSCHEMA:        GSCHEMA-1
  SEQUENCE:       ($part3 $part2)
  DIFF-SEQUENCE:  ($Diff-part2)
  PINSTS:         (PI-2 PI-1)
  POS-PINSTS:     (PI-2 PI-1)
  NEG-PINSTS:     NIL
  SUPERS:         NIL
  SUBS:           NIL
PSCHEMA-2 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 10 elements long.

PSCHEMA-1 is a $PSCHEMA
  INDEX:          1
  GSCHEMA:        GSCHEMA-1
  SEQUENCE:       ($part3 $part2)
  DIFF-SEQUENCE:  ($Diff-part1)
  PINSTS:         (PI-3 PI-1)
  POS-PINSTS:     (PI-1)
  NEG-PINSTS:     (PI-3)
  SUPERS:         NIL
  SUBS:           NIL
PSCHEMA-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 10 elements long.
NIL
(d *pinst*)
PI-3 is a PINST
  INDEX:          3
  PSCHEMAS:       (PSCHEMA-1)
  GINST:          GI-2
  SEQUENCE:       ([PART6] [PART5])
  DIFF-SEQUENCE:  (Diff-part([PART6],[PART5]))
```

```
PI-3 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 6 elements long.

PI-2 is a PINST
  INDEX:        2
  PSCHEMAS:     (PSCHEMA-2)
  GINST:        GI-2
  SEQUENCE:     ([PART5] [PART6])
  DIFF-SEQUENCE:  (Diff-part([PART5],[PART6]))
PI-2 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 6 elements long.

PI-1 is a PINST
  INDEX:        1
  PSCHEMAS:     (PSCHEMA-2 PSCHEMA-1)
  GINST:        GI-1
  SEQUENCE:     ([PART20] [PART21])
  DIFF-SEQUENCE:  (Diff-part([PART20],[PART21]))
PI-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 6 elements long.
NIL
(bracket-ex1) ;; Pose almost identical problem to bracket assembly
BRACKET1-ASSEMBLY
(apg bracket1)
(PINST PI-4)
(GINST GI-3)
[[PART45],[PART44]] < [[PART46],[PART44]]
EXECUTION-MONITOR
 Please input execution failure assertions justified by *execution-monitor*
 > bye
NIL
;; Note it does not query the user anymore with alternative
;; precedences because it learned not to apply pschema-1 from
;; previous experience
(dribble)
```

C.2. Learning Grouping Schema from Subassembly Precedence Violation

This section shows that after the user tells how to assemble bell assembly, the system remembers a generalized version of this scenario. Sub is an assembly structure with a subassembly. Sub1 is identical to sub except with different component names.

```
(bb)
NIL
(nomad-decl)
<NOM (LIAISON)>
```

```
(bell-ex)
BELL-ASSEMBLY
(describe bell) ;;bell is pointing to BELL assembly
BELL-ASSEMBLY is a ASSEMBLY
  NAME:          BELL
  PARTS:         ([PART1] [PART2] [PART3])
  LIAISONS:        (Liaison([PART2] [PART3]) Liaison([PART1]
[PART3]) Liaison([PART1] [PART2]))
  HIERARCHIES:     NIL ;; No hierarchies yet
BELL-ASSEMBLY is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 5 elements long.
BELL-ASSEMBLY
(apg bell) ;; generate assembly procedure for the bell assembly
>>Please choose one component for the base component
 Please choose the corresponding number:
 1: [PART1]
 2: [PART2]
 3: [PART3]
Enter the number of your choice (0 if nothing) > 1

>>Please choose satellite components
Please choose among the following objects by specifying their
corresponding numbers:
 1: [PART2]
 2: [PART3]
 0 if finished
 Choose a number> 1
 Choose a number> 2
Do you want to stop? (Y or N) Yes.
 Please choose in sequence
Please choose among the following objects by specifying their
corresponding numbers:
 1: [PART3]
 2: [PART2]
 0 if finished
 Choose a number> 2
 Choose a number> 1

;; PI-1 is the precedence used to create the assembly procedure
;; GI-1 is the grouping instance used to create the assembly procedure
(PINST PI-1)
(GINST GI-1)
[[PART2],[PART1]] < [[PART3],[PART1]]
EXECUTION-MONITOR
 Please input execution failure assertions justified by *execution-monitor*
  > bye ;; meaning execution of the assembly procedure is successful
NIL
(describe bell)
BELL-ASSEMBLY is a ASSEMBLY
  NAME:          BELL
```

```
  PARTS:        ([PART1] [PART2] [PART3])
  LIAISONS:     (Liaison([PART2] [PART3]) Liaison([PART1]
[PART3]) Liaison([PART1] [PART2]))
  HIERARCHIES:  (A1) ;; a hierarchy is constructed
BELL-ASSEMBLY is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 5 elements long.
BELL-ASSEMBLY
(d *hierarchy*)
A1 is a HIERARCHY
  INDEX:        1
  GINST:        GI-1
  BASE:         [PART1]
  SATELLITES:   ([PART3] [PART2])
  AUXLIAISONS:  NIL
  SUBHIERARCHIES: NIL
A1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.
(d *ginst*)
GI-1 is a GINST
  INDEX:        1
  GSCHEMAS:     (GSCHEMA-1)
  BASE:         [PART1]
  SATELLITES:   ([PART3] [PART2])
  LIAISONS:     (Liaison([PART1] [PART3]) Liaison([PART1] [PART2]))
  PINSTS:       (PI-1)
GI-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.
(d *pinst*)
PI-1 is a PINST
  INDEX:        1
  PSCHEMAS:     (PSCHEMA-2 PSCHEMA-1)
  GINST:        GI-1
  SEQUENCE:     ([PART2] [PART3])
  DIFF-SEQUENCE: (Diff-part([PART2],[PART3]))
PI-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 6 elements long.
(d *$gschema*)
GSCHEMA-1 is a $GSCHEMA
  INDEX:        1
  BASE:         $part1
  SATELLITES:   ($part2 $part3)
  LIAISONS:     ($Liaison2 $Liaison1)
  GINSTS:       ((GI-1 ($part1 . [PART1]) ($part3 .
[PART2]) ($part2 . [PART3]) ($Liaison1 . Liaison([PART1]
[PART3])) ($Liaison2 . Liaison([PART1] [PART2]))))
  POS-GINSTS:   (GI-1)
  NEG-GINSTS:   NIL
  PSCHEMAS:     (PSCHEMA-2 PSCHEMA-1)
  SUPERS:       NIL
  SUBS:         NIL
```

```
GSCHEMA-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 11 elements long.
NIL
;; Specifying new assembly structure called SUB assembly
(subassembly-ex)
SUB-ASSEMBLY
(describe sub)
SUB-ASSEMBLY is a ASSEMBLY
  NAME:          SUB
  PARTS:         ([PART7] [PART8] [PART9] [PART10])
  LIAISONS:        (Liaison([PART8] [PART10]) Liaison([PART8]
[PART9]) Liaison([PART7] [PART9]) Liaison([PART7] [PART8]))
  HIERARCHIES:    NIL
SUB-ASSEMBLY is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 5 elements long.
SUB-ASSEMBLY
(apg sub)
choose ginsts to choose
Please choose among the following objects by specifying their
corresponding numbers:
 1: GI-3
 2: GI-2
 0 if finished
 Choose a number> 2
 Choose a number> 0

(PINST PI-2)
(GINST GI-2)
[[PART8],[PART7]] < [[PART9],[PART7]]
[[PART8],[PART7]] < [[PART10],[PART8]]
[[PART9],[PART7]] < [[PART10],[PART8]]
EXECUTION-MONITOR
 Please input execution failure assertions justified by *execution-monitor*
  > (Failed-with ([[PART10],[PART8]] < [[PART8],[PART7]]))
  > bye
%Got nogoods:(E-7) ;; *exucution-monitor* detects a failure
NIL
(printenv *before-nogoods*) ;; List possible blames
E-7:* (EXECUTION-MONITOR, (GINST GI-2)) ;; GI-2 is to blame
NIL
(d *ginst*) ;; Display all the grouping instance in  memory
GI-3 is a GINST
  INDEX:         3
  GSCHEMAS:        (GSCHEMA-1)
  BASE:          [PART8]
  SATELLITES:      ([PART7] [PART10])
  LIAISONS:        (Liaison([PART7] [PART8]) Liaison([PART8] [PART10]))
  PINSTS:        (PI-3)
GI-3 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.
```

```
GI-2 is a GINST
  INDEX:        2
  GSCHEMAS:     (GSCHEMA-1)
  BASE:         [PART7]
  SATELLITES:   ([PART8] [PART9])
  LIAISONS:     (Liaison([PART7] [PART8]) Liaison([PART7] [PART9]))
  PINSTS:       (PI-2)
GI-2 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.

GI-1 is a GINST
  INDEX:        1
  GSCHEMAS:     (GSCHEMA-1)
  BASE:         [PART1]
  SATELLITES:   ([PART3] [PART2])
  LIAISONS:     (Liaison([PART1] [PART3]) Liaison([PART1] [PART2]))
  PINSTS:       (PI-1)
GI-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.
NIL
(d *pinst*) ;; Display all the precedences in memory
PI-3 is a PINST
  INDEX:        3
  PSCHEMAS:     (PSCHEMA-1)
  GINST:        GI-3
  SEQUENCE:     ([PART7] [PART10])
  DIFF-SEQUENCE:  (Diff-part([PART7],[PART10]))
PI-3 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 6 elements long.

PI-2 is a PINST
  INDEX:        2
  PSCHEMAS:     (PSCHEMA-2)
  GINST:        GI-2
  SEQUENCE:     ([PART8] [PART9])
  DIFF-SEQUENCE:  (Diff-part([PART8],[PART9]))
PI-2 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 6 elements long.

PI-1 is a PINST
  INDEX:        1
  PSCHEMAS:     (PSCHEMA-2 PSCHEMA-1)
  GINST:        GI-1
  SEQUENCE:     ([PART2] [PART3])
  DIFF-SEQUENCE:  (Diff-part([PART2],[PART3]))
PI-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 6 elements long.
NIL
(d *$gschema*)
;; GSCHEMA-2 is created because GSCHEMA-1 failed
```

```
GSCHEMA-2 is a $GSCHEMA
  INDEX:        2
  BASE:         $part1
  SATELLITES:     ($part2 $part3)
  LIAISONS:       ($Liaison1 $Liaison3)
  GINSTS:       NIL
  POS-GINSTS:     NIL
  NEG-GINSTS:     NIL
  PSCHEMAS:       NIL
  SUPERS:       NIL
  SUBS:         NIL
GSCHEMA-2 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 11 elements long.

;; GSCHEMA-1 will not be applied in the future because of GI-2
GSCHEMA-1 is a $GSCHEMA
  INDEX:        1
  BASE:         $part1
  SATELLITES:     ($part2 $part3)
  LIAISONS:       ($Liaison2 $Liaison1)
  GINSTS:       ((GI-3 ($Liaison1 . Liaison([PART8]
[PART10])) ($Liaison2 . Liaison([PART7] [PART8])) ($part3 .
[PART7]) ($part2 . [PART10]) ($part1 . [PART8])) (GI-2
($Liaison1 . Liaison([PART7] [PART9])) ($Liaison2 .
Liaison([PART7] [PART8])) ($part3 . [PART8]) ($part2 . [PART9])
($part1 . [PART7])) (GI-1 ($part1 . [PART1]) ($part3 .
[PART2]) ($part2 . [PART3]) ($Liaison1 . Liaison([PART1]
[PART3])) ($Liaison2 . Liaison([PART1] [PART2])))) .
  POS-GINSTS:     (GI-1)
  NEG-GINSTS:     (GI-2) ;; GI-2 is negative instance of GSCHEMA-1
  PSCHEMAS:       (PSCHEMA-2 PSCHEMA-1)
  SUPERS:       NIL
  SUBS:         NIL
GSCHEMA-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 11 elements long.
NIL
(describe sub)
SUB-ASSEMBLY is a ASSEMBLY
  NAME:         SUB
  PARTS:        ([PART7] [PART8] [PART9] [PART10])
  LIAISONS:       (Liaison([PART8] [PART10]) Liaison([PART8]
[PART9]) Liaison([PART7] [PART9]) Liaison([PART7] [PART8]))
  HIERARCHIES:    (A3) ;; A3 is the root of the hiearchy
SUB-ASSEMBLY is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 5 elements long.
SUB-ASSEMBLY
;; Here, hierarchies, A2 and A3, are used for SUB assembly
(d *hierarchy*)
;; After A2 is formed, A3 is created for the remaining objects
A3 is a HIERARCHY
```

```
  INDEX:        3
  GINST:        NIL
  BASE:         [PART8]
  SATELLITES:   ([PART10])
  AUXLIAISONS:  NIL
  SUBHIERARCHIES: (A2)
A3 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.

A2 is a HIERARCHY
  INDEX:        2
  GINST:        GI-2
  BASE:         [PART7]
  SATELLITES:   ([PART8] [PART9])
  AUXLIAISONS:  NIL
  SUBHIERARCHIES: NIL
A2 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.

A1 is a HIERARCHY
  INDEX:        1
  GINST:        GI-1
  BASE:         [PART1]
  SATELLITES:   ([PART3] [PART2])
  AUXLIAISONS:  NIL
  SUBHIERARCHIES: NIL
A1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.
NIL
;; New assembly, SUB1 that is almost identical to SUB
;; SUB1 and SUB are identical exept the names of parts.
(subassembly-ex1)
SUB1-ASSEMBLY
(describe sub1)
SUB1-ASSEMBLY is a ASSEMBLY
  NAME:         SUB1
  PARTS:        ([PART15] [PART16] [PART17] [PART18])
  LIAISONS:       (Liaison([PART16] [PART18])
Liaison([PART16] [PART17]) Liaison([PART15] [PART17])
Liaison([PART15] [PART16]))
  HIERARCHIES:  NIL
SUB1-ASSEMBLY is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 5 elements long.
SUB1-ASSEMBLY
(apg sub1)
;; Note, unlike the previous session where the gschema in
;; memory is applied, ask the user because the gschema
;; contains negative instance.
>>Please choose one component for the base component
 Please choose the corresponding number:
```

```
 1: [PART15]
 2: [PART16]
 3: [PART17]
 4: [PART18]
Enter the number of your choice (0 if nothing) > 2

>>Please choose satellite components
Please choose among the following objects by specifying their
corresponding numbers:
 1: [PART15]
 2: [PART17]
 3: [PART18]
 0 if finished
 Choose a number> 1
 Choose a number> 2
 Choose a number> 3
Do you want to stop? (Y or N) Yes.
 Please choose in sequence
Please choose among the following objects by specifying their corresponding numbers:
 1: [PART18]
 2: [PART17]
 3: [PART15]
 0 if finished
 Choose a number> 2
 Choose a number> 1
 Choose a number> 3
(PINST PI-4)
(GINST GI-4)
[[PART15],[PART16]] < [[PART17],[PART16]]
[[PART18],[PART16]] < [[PART15],[PART16]]
EXECUTION-MONITOR
 Please input execution failure assertions justified by *execution-monitor*
  > bye
NIL
*ginst*
(GI-4 GI-3 GI-2 GI-1)
(describe (car *ginst*))
GI-4 is a GINST
  INDEX:        4
  GSCHEMAS:     (GSCHEMA-3)
  BASE:         [PART16]
  SATELLITES:   ([PART18] [PART17] [PART15])
  LIAISONS:     (Liaison([PART16] [PART18])
Liaison([PART16] [PART17]) Liaison([PART15] [PART16]))
  PINSTS:       (PI-4)
GI-4 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.
GI-4
(describe (car *$gschema*))
GSCHEMA-3 is a $GSCHEMA
```

```
  INDEX:        3
  BASE:         $part4
  SATELLITES:     ($part5 $part6 $part7)
  LIAISONS:       ($Liaison6 $Liaison5 $Liaison4)
  GINSTS:       ((GI-4 ($part4 . [PART16]) ($part7 .
[PART15]) ($part6 . [PART17]) ($part5 . [PART18]) ($Liaison4 .
Liaison([PART16] [PART18])) ($Liaison5 . Liaison([PART16]
[PART17])) ($Liaison6 . Liaison([PART15] [PART16]))))
  POS-GINSTS:     (GI-4)
  NEG-GINSTS:     NIL
  PSCHEMAS:       (PSCHEMA-4 PSCHEMA-3)
  SUPERS:       NIL
  SUBS:         NIL
GSCHEMA-3 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 11 elements long.
GSCHEMA-3
(dribble)
```

C.3. Specializing Grouping Schema

This session shows learning substrantilly differenct grouping knowledge.

```
(bb)
NIL
(nomad-decl)
<NOM (LIAISON)>
(bell-ex1) ;; defining BELL1 assembly
BELL1-ASSEMBLY
(apg bell1)
>>Please choose one component for the base component
 Please choose the corresponding number:
 1: [PART19]
 2: [PART20]
 3: [PART21]
Enter the number of your choice (0 if nothing) > 1

>>Please choose satellite components
Please choose among the following objects by specifying their
corresponding numbers:
 1: [PART20]
 2: [PART21]
 0 if finished
 Choose a number> 1
 Choose a number> 2
Do you want to stop? (Y or N) Yes.
 Please choose in sequence
Please choose among the following objects by specifying their
corresponding numbers:
```

```
1: [PART21]
2: [PART20]
0 if finished
Choose a number> 2
Choose a number> 1
(PINST PI-1)
(GINST GI-1)
[[PART20],[PART19]] < [[PART21],[PART19]]
EXECUTION-MONITOR
Please input execution failure assertions justified by *execution-monitor*
 > bye
NIL
(d *ginst*)
GI-1 is a GINST
  INDEX:        1
  GSCHEMAS:       (GSCHEMA-1)
  BASE:         [PART19]
  SATELLITES:     ([PART21] [PART20])
  LIAISONS:       (Liaison([PART19] [PART21]) Liaison([PART19] [PART20]))
  PINSTS:       (PI-1)
GI-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.
NIL
(4parts) ;; defining a new assembly: FOUR
FOUR-ASSEMBLY
(describe four)
FOUR-ASSEMBLY is a ASSEMBLY
  NAME:         FOUR
  PARTS:        ([PART11] [PART12] [PART13] [PART14])
  LIAISONS:       (Liaison([PART13] [PART14])
Liaison([PART12] [PART13]) Liaison([PART11] [PART14])
Liaison([PART11] [PART13]) Liaison([PART11] [PART12]))
  HIERARCHIES:    NIL
FOUR-ASSEMBLY is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 5 elements long.
FOUR-ASSEMBLY
(apg four) ;; Generate assembly procedure
(PINST PI-5)
(GINST GI-5) ;; GI-5 and PI-5 are used to create the precedences
[[PART13],[PART11]] < [[PART14],[PART11]]
[[PART12],[PART11]] < [[PART13],[PART11]]
EXECUTION-MONITOR
Please input execution failure assertions justified by *execution-monitor*
 > bye
NIL
(d *ginst*)
;; GI-5 is a collapsed structure from GI-2 and GI-4
GI-5 is a GINST
  INDEX:        5
  GSCHEMAS:       (GSCHEMA-2) ;; New schema is created
```

```
  BASE:         [PART11]
  SATELLITES:     ([PART12] [PART13] [PART14]) ;; Three satellites
  LIAISONS:       (Liaison([PART11] [PART12])
Liaison([PART11] [PART13]) Liaison([PART11] [PART14]))
  PINSTS:       (PI-5)
GI-5 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.

GI-4 is a GINST
  INDEX:        4
  GSCHEMAS:     (GSCHEMA-1)
  BASE:         [PART11]
  SATELLITES:     ([PART13] [PART14])
  LIAISONS:       (Liaison([PART11] [PART13]) Liaison([PART11] [PART14]))
  PINSTS:       (PI-4)
GI-4 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.

GI-3 is a GINST
  INDEX:        3
  GSCHEMAS:     (GSCHEMA-1)
  BASE:         [PART11]
  SATELLITES:     ([PART12] [PART14])
  LIAISONS:       (Liaison([PART11] [PART12]) Liaison([PART11] [PART14]))
  PINSTS:       (PI-3)
GI-3 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.

GI-2 is a GINST
  INDEX:        2
  GSCHEMAS:     (GSCHEMA-1)
  BASE:         [PART11]
  SATELLITES:     ([PART12] [PART13])
  LIAISONS:       (Liaison([PART11] [PART12]) Liaison([PART11] [PART13]))
  PINSTS:       (PI-2)
GI-2 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.

GI-1 is a GINST
  INDEX:        1
  GSCHEMAS:     (GSCHEMA-1)
  BASE:         [PART19]
  SATELLITES:     ([PART21] [PART20])
  LIAISONS:       (Liaison([PART19] [PART21]) Liaison([PART19] [PART20]))
  PINSTS:       (PI-1)
GI-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 7 elements long.
NIL
(d *$gschema*)
GSCHEMA-2 is a $GSCHEMA
```

```
  INDEX:        2
  BASE:         $part4
  SATELLITES:     ($part5 $part6 $part7)
  LIAISONS:       ($Liaison5 $Liaison4 $Liaison3)
  GINSTS:       ((GI-5 ($part4 . [PART11]) ($part7 . [PART14])
($part6 . [PART13]) ($part5 . [PART12]) ($Liaison3 . Liaison([PART11] [PART12]))
($Liaison4 . Liaison([PART11] [PART13])) ($Liaison5 . Liaison([PART11] [PART14]))))
  POS-GINSTS:     (GI-5)
  NEG-GINSTS:     NIL
  PSCHEMAS:       (PSCHEMA-4 PSCHEMA-3)
  SUPERS:       NIL
  SUBS:         NIL
GSCHEMA-2 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 11 elements long.

GSCHEMA-1 is a $GSCHEMA
  INDEX:        1
  BASE:         $part1
  SATELLITES:     ($part2 $part3)
  LIAISONS:       ($Liaison2 $Liaison1)
  GINSTS:       ((GI-4 ($Liaison1 . Liaison([PART11] [PART14]))
($Liaison2 . Liaison([PART11] [PART13])) ($part3 . [PART13])
($part2 . [PART14]) ($part1 . [PART11]))
(GI-3 ($Liaison1 . Liaison([PART11] [PART14]))
($Liaison2 . Liaison([PART11] [PART12])) ($part3 . [PART12])
($part2 . [PART14]) ($part1 . [PART11]))
(GI-2 ($Liaison1 . Liaison([PART11] [PART13]))
($Liaison2 . Liaison([PART11] [PART12])) ($part3 . [PART12])
($part2 . [PART13]) ($part1 . [PART11]))
(GI-1 ($part1 . [PART19]) ($part3 . [PART20]) ($part2 . [PART21])
($Liaison1 . Liaison([PART19] [PART21])) ($Liaison2 . Liaison([PART19] [PART20]))))
  POS-GINSTS:     (GI-1)
  NEG-GINSTS:     NIL
  PSCHEMAS:       (PSCHEMA-2 PSCHEMA-1)
  SUPERS:       NIL
  SUBS:         NIL
GSCHEMA-1 is implemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 11 elements long.
NIL
(dribble)
```

REFERENCES

(1) Allen, J. F., "Maintaining Knowledge about Temporal Intervals,"*Communications of the ACM*, Volume 26, No. 11, pp. 832-843, November 1983.

(2) Allen, J. F., "Towards a general Model of Action and Time," Artificial Intelligence 23 (2), July, 1984.

(3) Allen, J. F. and Kooman J. A., "Planning Using a Temporal World State," Proceeding of IJCAI, 1983.

(4) Bledsoe, W. W., "The SUP-INF Method in Presberger Arithemetic," Memo ATP-18, Math. Dept. U of Texas, 1974.

(5) Boyer, R. S. and Moore J. S., "A Fast String Searching Algorithms," Communication of the ACM, Vol 20, no 10. pp. 762-772, Oct. 1977.

(6) Brooks, R.A. and Lozano-Perez, T., "A Subdivision Algorithm in Configuration Space for Findpath with Rotation," Proceedings of the Eighth International Joint Conference on Artificial Intelligence, pp. 799-806, August 1983.

(7) Baumgart, B. G., "Geometric Modeling for Computer Vision," PhD thesis, Stanford University, August 1974.

(8) Chapman, D., "Planning for Conjunctive Goals," MS thesis, MIT, 1985.

(9) Chen, K., "An Inductive Engien for the Acquisition of Temporal Knowledge," PhD thesis, Department of Computer Science, University of Illinois, Urbana, 1988.

(10) Denavit, J. and Hartenberg, R. S., "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," Journal of Applied Mechanics Transaction ASME vol. 77, pp 215-221, 1955.

(11) Donald, B. R., "Motion Planning with Six Degrees of Freedom," TR-791, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1984.

(12) Hartenberg R. S. and Denavit, J., "Kinematic Synthesis of Linkages," McGraw-Hill Book, 1964.

(13) DeJong, G. and Mooney, R., "Explanation-Based Learning: An Alternative View," Machine Learning Journal, 1986.

(14) Fahlman, S., "A Planning System for Robot Construction Tasks," Artificial Intelligence, vol 5, pp. 1-49, 1974.

(15) de Kleer, J., "How Circuits Work," Artificial Intelligence, vol. 24, pp 205-280, 1984.

(16) de Kleer, J., "An Assumption-Based Truth Maintenance System," Artificial Intelligence, vol. 28, no. 1, 1986.

(17) De Fazio T. L. and Whitney D. E., "Simplified Generation of All Mechanical Assembly Sequences," IEEE Journal of Robotics and Automation, 1988.

(18) Fikes, R. E. and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, vol. 2, pp. 189-208, 1971.

(19) Fikes, R. E., Hart, P. E. and Nilsson, N.J, "Learning and Executing Generalized Robot Plans," Artificial Intelligence, vol 3, pp 251-288, 1972.

(20) Forbus, K., "Qualitative Process Theory," Artificial Intelligence, vol. 24, pp. 85-168, 1984.

(21) Haussler, D., "Learning Conjunctive Concepts in Structural Domains," UCSC-CRL-87-1, University of California at Sata Cruz, February 8, 1987.

(22) Hayes, P. J., "The Naive Physics Manifesto," in Expert Systems in the Micro-Electronic Age, D. Michie (ed), Edinburgh University Press, May 1979.

(23) Hayes-Roth, F., "Using Proofs and Refutations to Learn from Experience," in Machine Learning, An Artificial Intelligence Approach, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (eds), Tioga, Palo Alto, CA, 1983.

(24) Hendrix, G. G., "Modeling Simultaneous Actions and Continuous Processes", Artificial Intelligence 4 (1973),145-180.

(25) Homem de Mello, L. S. and Sanderson, A C., "AND/OR Graph Representation of Assembly Plans" AAAI-86, 1113-1119.

(26) Hunt, Kenneth Henderson, "Kinematic Geometry of Mechanisms," Oxford University Press, 1978.

(27) Malcolm, C. A. and Fothegill, A. P., "Some Architectural Implications of the Use of Sensors," Department of Artificial Intelligence Unversity of Ediburgh Research Paper No. 294, October 1986.

(28) Kodratoff, Y. and Michalski, R.S. (eds), "Machine Learning: An Aritificial Intellitence Approach," Volume III, Morgan Kaufmann, forthcoming 1989.

(29) Ko, H. and Lee, K., "Automatic Assembling Procedure Generation from Mating Conditions," Computer Aided Design, vol. 19, num 1, pp. 3-10, Jan/Feb 1987.

(30) Larson, J. B., "Inductive Inference in Variable-Valued Predicate Logic System VL_{21}: a Methodology and Computer Implementation," PhD thesis, Report No. 869, Department of Computer Science, University of Illinois, May, 1977.

(31) Lee, K. and Gossard, D. C., "A Hierarchical Data Structure for Representing Assemblies: Part 1," Computer Aided Design, vol. 17, no 1, pp. 15-19, Jan/Feb 1985.

(32) Lee, K. and Andrew, G., "Inference of the Positions of Components in an Assembly: Part 2," Computer Aided Design, vol. 17, no 1, pp. 20-24, Jan/Feb 1985.

(33) Lieberman, L. I. and Wesley, M. A., "Autopass: An Automatic Program System for Computer Controlled Mechanical Assembly," IBM J. RES. DEV, pp. 321-333, July 1977.

(34) Lozano-Perez, T., "Robot Programming," A. I. Memo No. 698, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1982.

(35) Lozano-Perez, T., "Task Planning," Chapter in Robot Motion: Planning and Control, M. Brady (eds), MIT Press, 1983.

(36) Mason, M. T., "Compliance and Force Control for Computer Controlled Manipulators," TR-515, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, April, 1979.

(37) McDermott, D., "Temporal Logic for Reasoning About Processes and Plans," Cognitive Science, vol. 6, pp. 101-155, 1982.

(38) Michalski, R. S., "A Theory and Methodology of Inductive Learning," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, T. M. Mitchell (eds.), Tioga Publishing Co., 1983.

(39) Michalski, R. S., "Concept Learning," in the Encyclopedia of Artificial Intelligence, John Wiley & Sons, E. Shapiro (eds), January 1987.

(40) Michalski, R. and Ko, H, "On the Nature of Explanation: Why Wine Bottle Shattered?", AAAI 1988 Spring Symposium Series, Mini-Symposium: Explanation-Based Learning, March 1988.

(41) Michalski, R. S., Ko, H. and Chen, K, "Qualitative Prediction: The SPARC/G Methodology for Describing and Predicting Discrete Processes," in **Expert Systems**, P. Dufour and A. Van Lamsweerde (eds), Academic Press Incorporated, pp 125-158, 1987.

(42) Mitchell, T. M., Keller, R. M. and Kedar-Cabelli S. T., "Explanation-Based Generalization: A unifying View," in *Machine Learning Journal*, vol 1, January 1986.

(43) Mortesson, M. E., "Geometric Modeling," John Wiley and Sons, New York, 1985.

(44) Mostow, D. J., "Machine Transformation of Advice into a Heuristic Search Procedure" in Machine Learning, An Artificial Intelligence Approach, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (eds), Tioga, Palo Alto, CA, 1983.

(45) Nilsson, N. J., "Principles of Artificial Intelligence," Tioga, Palo Alto, CA, 1980.

(46) Paul, R.P., "Robot Programming," MIT Press, 1982.

(47) Poplestone, R. J., Ambler, A. P., and Bellos, I. M., "An Interpreter for a Language for Describing Assemblies," Artificial Intelligence Journal, vol 14, pp 79-107, 1980.

(48) Prenting, T.O. and Battaglin, R.M., "The Precedence Diagram: A Tool for Analysis in Assembly Line Balancing," J. Ind. Eng., Vol 15, No. 4, pp. 208-213, July-August 1964.

(49) Quinlain, R. J., "Learning Efficient Classification Procedures and their Application to Chess End Games," in Machine Learning, An Artificial Intelligence Approach, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (eds), Tioga, Palo Alto, CA, 1983.

(50) Requicha A.A.G. and Voelcker, H.B., "Solid Modeling: Current Status and Research Direction," Institute of Electrical and Electronics Engineers Computer Graphics and Applications 3, pp. 25-37, Oct 1983.

(51) Reuleaux, F., "Kinematics of Machinery," edited and translated by Alex B. W. Kennedy, London Macmillan and co., 1876.

(52) Sacerdoti, E.D., "Planning in a Hierarchy of Abstraction Spaces," Artificial Intelligence, 5 (2), pp. 115-135, 1974.

(53) Schoppers, M. J., "Universal Plans for Reactive Robots in Unpredictable Environment," Proc 10th IJCAI, 1987.

(54) Segre, A. M., "Explanation-Based Learning of Generalized Robot Assembly Plans," PhD thesis, UILU-ENG-87-2208, Coordinated Science Laboratory, Univeristy of Illinois, January 1987.

(55) Smithers, T. and Malcolm, C., "A Behavioral Approach to Robot Task Planning and Off-line Programming," DAI Research Paper No. 306, May 1987.

(56) Stepp, R. E., "Conjunctive Conceptual Clustering: A Methodology and Experimentation," PhD thesis, UIUCDCS-R-84-1189, Department of Computer Science, University of Illinois, November 1984.

(57) Sussman, G. J., "Computer Model of Skill Acquisition," American Elsevier Publishing Company, Inc., New York, 1975.

(58) Ulrich, K.T. and Seering W.P., "Function Sharing in Mechanical Design," draft, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1988.

(59) Waldinger, R., "Achieving Several Goals Simultaneously," in Machine Intelligence 8, pp94-136.

(60) Wesley, M.A., Lozano-Perez, T., Lieberman, L.I., Lavin, M.A., and Grossman, C.D., "A Geometric Modeling System for Automated Mechanical Assembly," IBM J. Res. Dev., Vol. 24, No. 1, pp. 64-74, January 1980.

(61) Winston, P. H., "Learning Structural Descriptions from Examples," PhD thesis, MAC TR-76, Massachusetts Institute of Technology, September, 1970.