FINAL TECHNICAL REPORT PFTR-1027-76-8
August 1976

Contract No. 954380

STUDY AND DEVELOPMENT OF TECHNIQUES FOR AUTOMATIC CONTROL OF REMOTE MANIPULATORS

Efraim Shaket
Antonio Leal

Prepared For:

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, California 91103

This work was performed for the Jet Propulsion Laboratory, California Institute of Technology, sponsored by the National Aeronautics and Space Administration under Contract NAS7-100.

PERCEPTRONICS

6271 VARIEL AVENUE • WOODLAND HILLS • CALIFORNIA 91364 • PHONE (213) 884-7470

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION AND BACKGROUND

1.1 General

This is the final report on an overall conceptual design for an autonomous control system of remote manipulators which utilizes feedback. The system consists of a description of the high-level capabilities of a model from which design algorithms can be constructed. For the current remote manipulator system, the design goal is:

> To perform simple remote manipulation tasks in a partially unknown environment without human assistance.

The autonomous capability is achieved through automatic planning and locally controlled execution of the plans. The operator gives his commands in high level task-oriented terms. The system transforms these commands into a plan -- a sequence of detailed low-level commands. It uses built-in procedural knowledge of the problem domain and an internal model of the current state of the world. The plans include mechanisms to control execution using information collected from input sensors. They are also capable of recovering from execution problems by building alternative subplans. The following are samples of primitive commands into which a high-level operator command is transformed:

(1) Open and close jaws.
(2) Control each link of the manipulator.
(3) Orient hand position by wrist movement.
(4) Move hand to a predesignated position (coordinate in 3-space).

Figure 1 shows a typical manipulator configuration with four proximity sensors capable of detecting nearby objects.

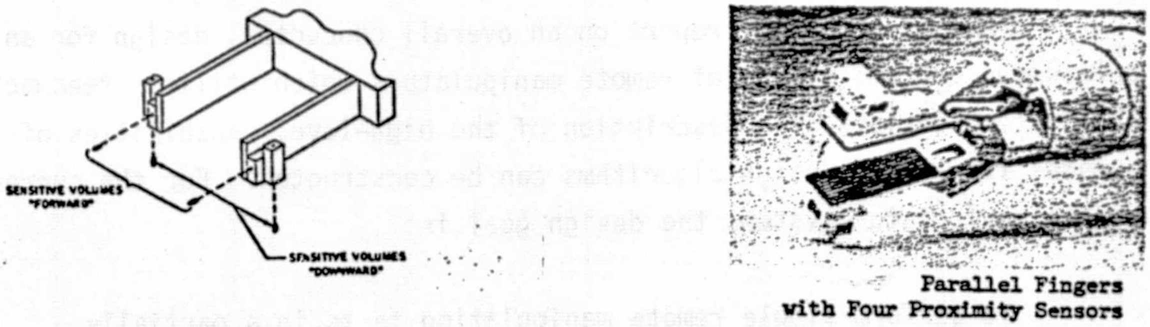

SENSITIVE VOLUMES "FORWARD"
SENSITIVE VOLUMES "DOWNWARD"
Parallel Fingers with Four Proximity Sensors

FIGURE 1. MANIPULATOR HAND WITH SENSORS

In addition, information can be obtained that specifies the current state of the arm and hand. For example, the following information will be considered to be available at any time:

(1) Current location, speed, and force in each link.
(2) The global current location of the hand.
(3) Hand orientation (including twist).
(4) Current values of all sensor readings.
(5) Inward force being applied to jaws.
(6) Vector of forces at the wrist.

The data processing requirements of the final system will be limited to the memory and processing capability found on current mini-computers. That is, the algorithms will eventually be locally implemented.

1.2 State Space Models

Most of the problem solving systems developed by research in Artificial Intelligence (AI) are based on some variation of the state space model. A problem presented in this formalization consists of an initial state, a set of possible subsequent states, and a set of possible actions, together with a specification of how the various states can be produced from each other by different actions. A solution to a state space problem is any sequence of actions that leads from the initial state to the desired "goal" state and avoids undesired states (Nilsson, 1971).

Essentially, the algorithms devised to solve problems formulated in the state space model are graph searching algorithms. By representing the state space model as a graph with states as nodes and actions as arcs, the algorithms can expand the nodes of the graph in some order by applying all applicable operators to each node in turn. When a goal node is encountered, the path to it from the starting state is retraced and is given as the solution. The order of node expansion in these systems is determined by increasingly complex heuristic methods, such as static state evaluation functions, dynamic ordering, various pruning methods, or combinations of these.

The search space formulation for any significant part of a real world problem proves to be much too large for a successful application of these problem solving approaches. This weakness can be attributed to the required uniform application of operators to states thus limiting the number of operators that can be considered at each step to a very small number. Furthermore, the problem specific information -- the heuristics -- is incidental to the underlying blind search mechanism. Additional problems with the model are the discrete nature of the modeling of world states and time, and its incapability to consider events influenced by decisions done by processes outside the searching algorithm.

1.3 Theorem Proving Methods

Building on an analogy between the processes of proving theorems and problem solving, later systems tried to use automatic resolution algorithms to solve problems. In these systems the world model is represented as a set of well formed formulas (wffs) of the first order predicate calculus. Operators are defined in terms of preconditions which must be satisfied in a given world model for the operator to be applicable there, as well as a set of "add" and "delete" wffs which specify the changes to the world model accomplished by applying the operator. STRIPS (Fikes, 1971) is the best known system employing this formulation. It uses means-ends analysis as in GPS (Ernst and Newell, 1969) to find which operators are relevant to reducing the "difference" between the current world model and the desired goal. It uses resolution theorem proving to test the applicability of the relevant operators in a given world model.

Experience with STRIPS has shown that although a powerful heuristic had been added to the search procedure, the system is bogged down by the resolution proof algorithm when it is applied to the large unstructured set of wffs representing a world model. The system was not successful in finding a solution composite operator when more than about ten steps were needed in the solution. Also, the solution time was an exponential function of the length of the solution path.

1.4 Search in Abstraction Space

A later version of this approach, ABSTRIPS (Sacerdoti 1973) has achieved a significant reduction in the amount of search performed by the system by conducting the search in a hierarchy of problem spaces at various levels of abstractions. A very crude interpretation of the concept of "levels of abstraction" is adopted here. The preconditions of the various operators are sorted and taged according to an estimate of their importance. Starting at a high level of abstraction, only the most important preconditions are considered by the search algorithm. Thus, the amount of detail that the system has to consider is reduced considerably and it can find a sequence of important subgoals leading from the starting state to the goal. Subsequently, the system searches for paths between these "island states" considering the operators in more and more detail. ABSTRIPS was successful in finding solution paths containing up to two hundred steps and the search effort increased much slower as a function of solution length.

1.5 Planning and the Procedural Net

The problem solving systems discussed above employ some general mechanism which is applied uniformly to a formulation of domain specific information. This generality is paid for by reducing the complexity of the problems that such a system can solve. Still, a substantial amount of domain-specific information has to be incorporated into the states, the operators and the relations between them. Rather than add domain-specific information as an afterthought, Finkel, et al (1974) adopted a more direct programming approach. His system, called AL, is a programming system for developing specifications of tasks for industrial manipulators.

An interesting planning mechanism was proposed by Sacerdoti (1975) who used a construct which is called a procedural net. The system was developed as a mechanism to give advice to a human apprentice. The procedural net is a hierarchically organized combination of data structures and procedures from which step-by-step directions to accomplish a given task can be easily extracted. The net can generate the instructions at various levels of detail depending on the sophistication of the apprentice. It has the capability of controlling the execution of a plan and recovering from errors.

1.6 Sensor Utilization

A significant part of Artificial Intelligence research on sensory input has concentrated on visual sensors, that is, sensing the environment by some form of television camera and using the acquired information to construct an internal symbolic model of the scene. This approach involves scene analysis techniques and complex algorithms for machine perception. Furthermore, the more successful systems to date achieved reasonable success only in a scene which included polyhedron objects where clear edges are used as clues for segmenting the scene into separate objects. Analysis of real image data achieved much less success (Shirai 1975) with well lit complex objects and (Zucker 1975) describing region growing. This can be attributed to the lack of appropriate mechanisms to represent a real scene which usually includes irregular objects. Although ultimately functionally optimum, as it provides a gross view plus local details of the environment, visual perception is premature both theoretically and practically since it requires complicated hardware as well as complex software structure.

Direct sensing of the environment by a manipulator through some set of sensors, and using this information in a local execution control program, offers a more immediate and practical solution to the problem of utilizing sensory information for the control of motion in a real environment. With only local sensory information, however, such as tactile, force, or proximity, the manipulator can obtain data for an internal model when planning global manipulations.

An analogy is commonly drawn between the data associated with a program in a computing system and the data obtained by sensors in the real environment of a manipulator control system. There are, however, significant differences which impose substantial changes in attitude and organization on the system which interprets and uses the sensor data. Program data is defined by the programmer and is available in simple known structures (such as vectors, arrays or lists) to best suit the algorithm which uses it. The amount of data that is generated or obtained by a program is determined by how much is needed and can be used. In most cases all the data needed for a program is accessible (within the access structure of the programming language). When an instruction is given in a programming language, its correct, well-defined execution is assumed. With sensory data, on the other hand, only the superficial aspects of the data are known in advance. It is highly redundant and contains hidden complex structures. These are not immediately apparent in the local sensory level -- the perception process must impose a global structure on the observational data to render it meaningful. The amount of data obtainable from a real scene by various sensor aparatus is prohibitive. Data reduction is mandatory and the important issues are what information is relevant, where it can be obtained, and how it should be used, all within the resources available. The sensory apparatus available to the system can never be complete or perfectly accurate. The world picture so obtained must always be a distorted partial truth. Consequently, the sensory information must be considered suspicious until properly verified by several sources. On the output side, a control command cannot be assumed executed until some evidence is available to verify it.

The system described in this report, which uses sensory information to control the motions of the manipulator, is an initial attempt to address these issues. The mechanism proposed is an application of the "TOTE" (Test-Operate-Test-Exit) unit developed by Miller, et al (1960). This type of loop control will be described in detail in a subsequent section.

2. SYSTEM OVERVIEW

2.1 Global System Organization

The essential function of the system, as can be seen in the block diagram in Figure 2, is to transform task oriented commands given by the operator into an explicit, detailed command sequence compatible with the manipulator hardware. The information transmitted between the blocks can be viewed as a hierarchy of languages with an appropriate processor translating from one stage to the next. Going from left to right in the figure, the languages become more specific and detailed. Each processor considers its input as commands from its predecessor, using problem-domain knowledge from the world model to construct expressions in a lower level language. These, in turn, are commands for the next stage. The three blocks in the figure essentially perform the following functions. The fuzzy language translator accepts high level commands in a fuzzy, task oriented user-compatible command language. The planner and goal oriented execution controller performs planning and transforms a general, high level command, using the specific state of the environment, into a detailed sequence of specific commands for the manipulator. The low level controller utilizes the information from various sensors to control the execution of the detailed plans generated for it.

The world model contains procedural information required at the various translation levels. To aid in translating fuzzy commands to semantic structures, the world model contains information about the specific information necessary to instantiate an incomplete command. It also contains procedures for filling in missing information according to the current and expected state of the environment. For translating from semantic structures to primitive commands, the world model has a collection of hierarchial skeletal plans which embody the domain specific planning knowledge. These procedures are used in expanding a high level command into a detailed plan. Finally,

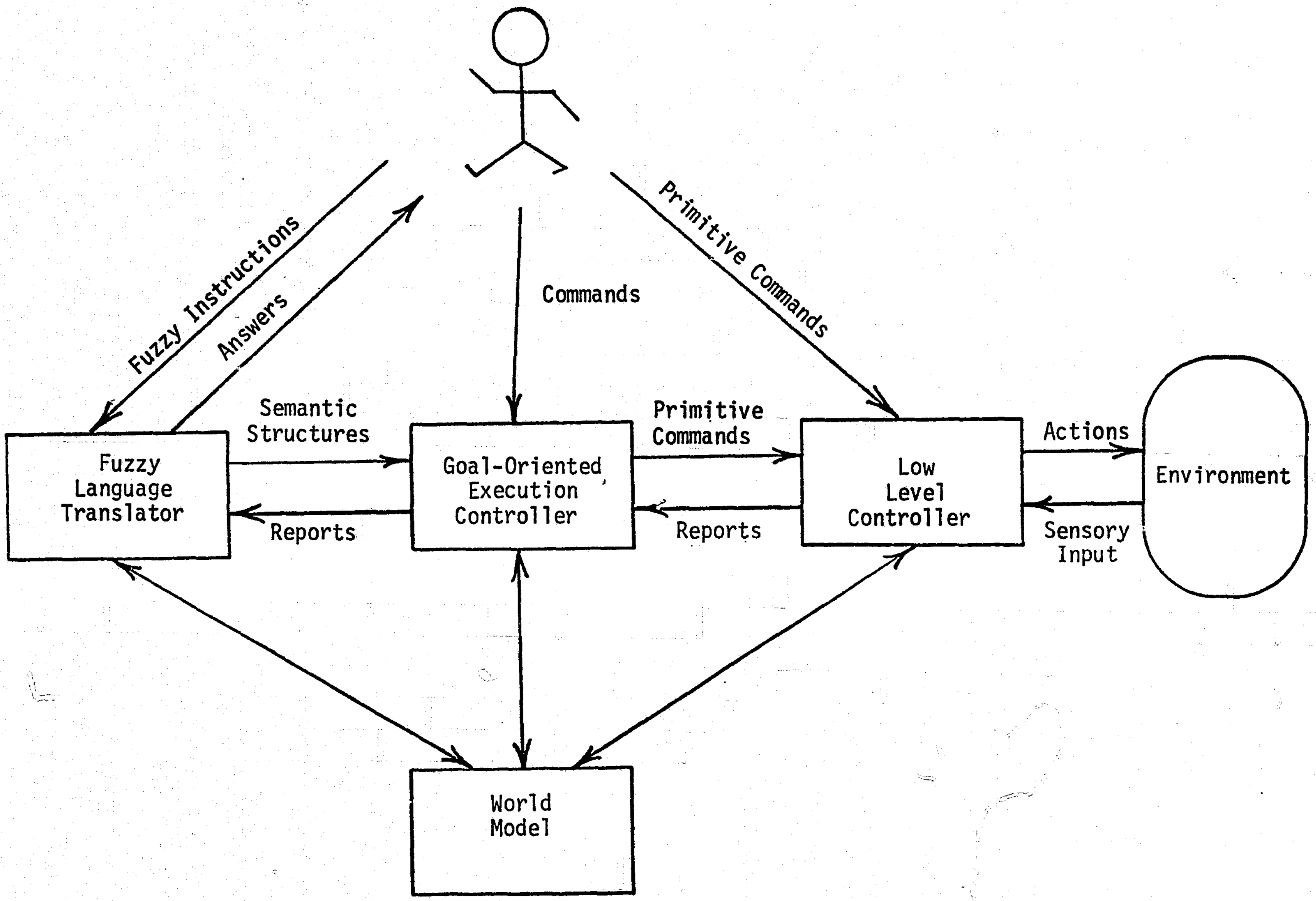


FIGURE 2. SYSTEM BLOCK DIAGRAM

it contains an explicit updated model of the world which is maintained while the manipulator actions are taking place. It is capable of monitoring continuous processes in the environment which change without the manipulator's initiation or outside of the immediate scope of its sensors.

2.2 Low Level Control

The low level subsystem in Figure 3 is organized around primitive control commands which contain two parts: (a) an incremental command (with parameters) and (b) termination and continuation conditions. When a command is given to the "Move Increment Calculator", an incremental command for the manipulator links is generated. The "Arm Monitor" compares the signals going outward to the manipulator with the position and speed sensors at the links and keeps track of the manipulator current position. "Pending Sensory Conditions" contains various monitors, established globally or with the current primitive command, that watch for particular events in the world model, problems in the manipulator motions, or particular patterns in the sensory information. When such an event occurs, an interrupt is issued to stop the incremental motion and to the "Execution Control" (Figure 2) which will decide on the next move. Finkel (1974) controlled primitive commands with similar sensory dependent termination conditions.

2.3 Planning and Execution Control

This subsystem (Figure 4) contains two interpreters: one for generating plans from the given commands and the other for monitoring the execution of these plans. Plans are expressed in the system as a hierarchical collection of skeleton plans in a special programming language. The primitives of this language are operators and processes useful in developing a detailed plan from a high level command.

The planning interpreter, (Planner), accepts commands either from the operator or from semantic structures from the fuzzy language translator.

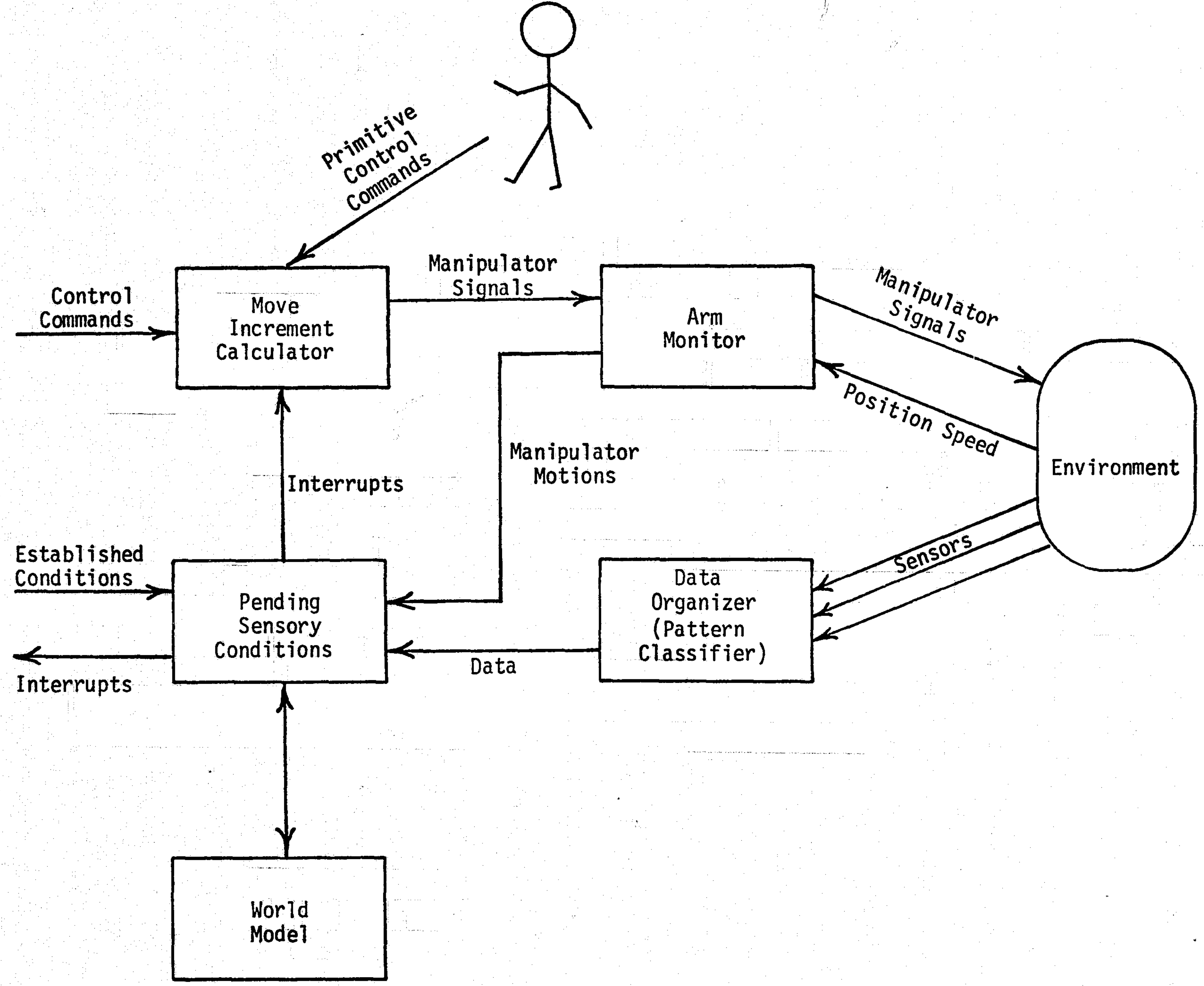


FIGURE 3. LOW LEVEL SUBSYSTEM

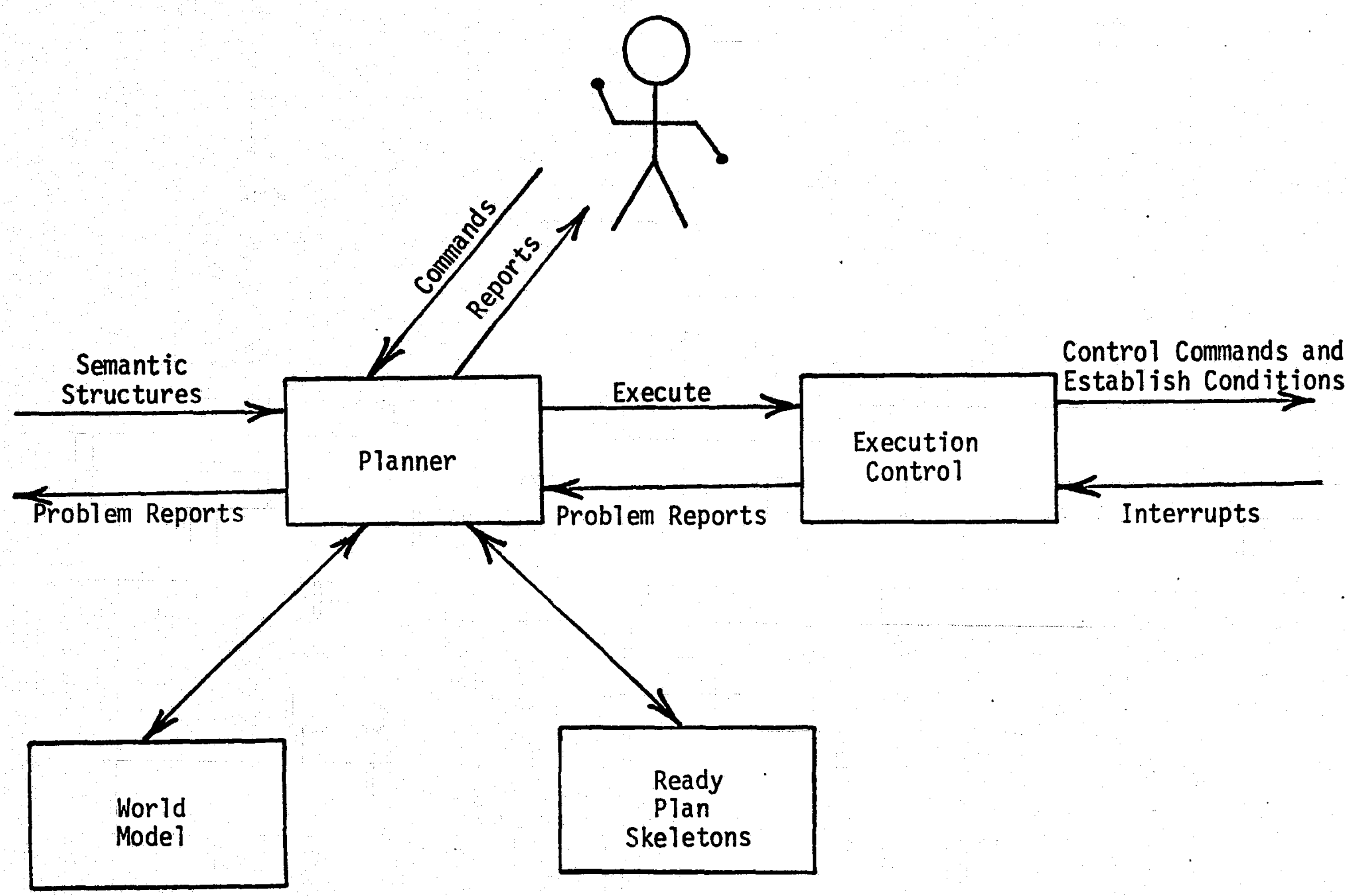


FIGURE 4. PLANNING AND EXECUTION CONTROLLER

Guided by the domain-specific knowledge contained in the hierarcy of the skeleton plan, it constructs the procedural net corresponding to the given command. The procedural net is a combination of the data and procedures that is expanded "top-down" in a breadth first order and represents in a hierarchical manner the sequence of actions that must take place in order to accomplish the high level command in the current setting. The procedural net, when developed down to primitive actions, is given to the "Execution Control" for execution.

The interaction between the Planner and the Execution Controller is two-directional to allow for two capabilities: (1) plan-time execution -- a surveillance motion which is actually carried out during planning to obtain crucial information needed for the planning process and (2) execution-time planning -- actually interrupting the execution to perform required planning.

2.4 Fuzzy Language Translator

This subsystem (see Figure 5) provides natural communication with the user. It accepts as input fuzzy instructions in a language compatible with the user. It contains morphonic, syntactic, and semantic knowledge about the problem domain in general and the specific state of the current environment to properly interpret the user's instructions. Semantic structures are produced which are commands compatible with the planner. This subsystem is similar to the system described by Winograd (1972) modified to handle the fuzzy concepts common in natural lnaguage. The Answer/Inquiry Generator provides communication in the opposite direction,to ask the user for clarifications of ambiguous commands to produce answers to his inquiries about the state of the world or any internal state of the system.

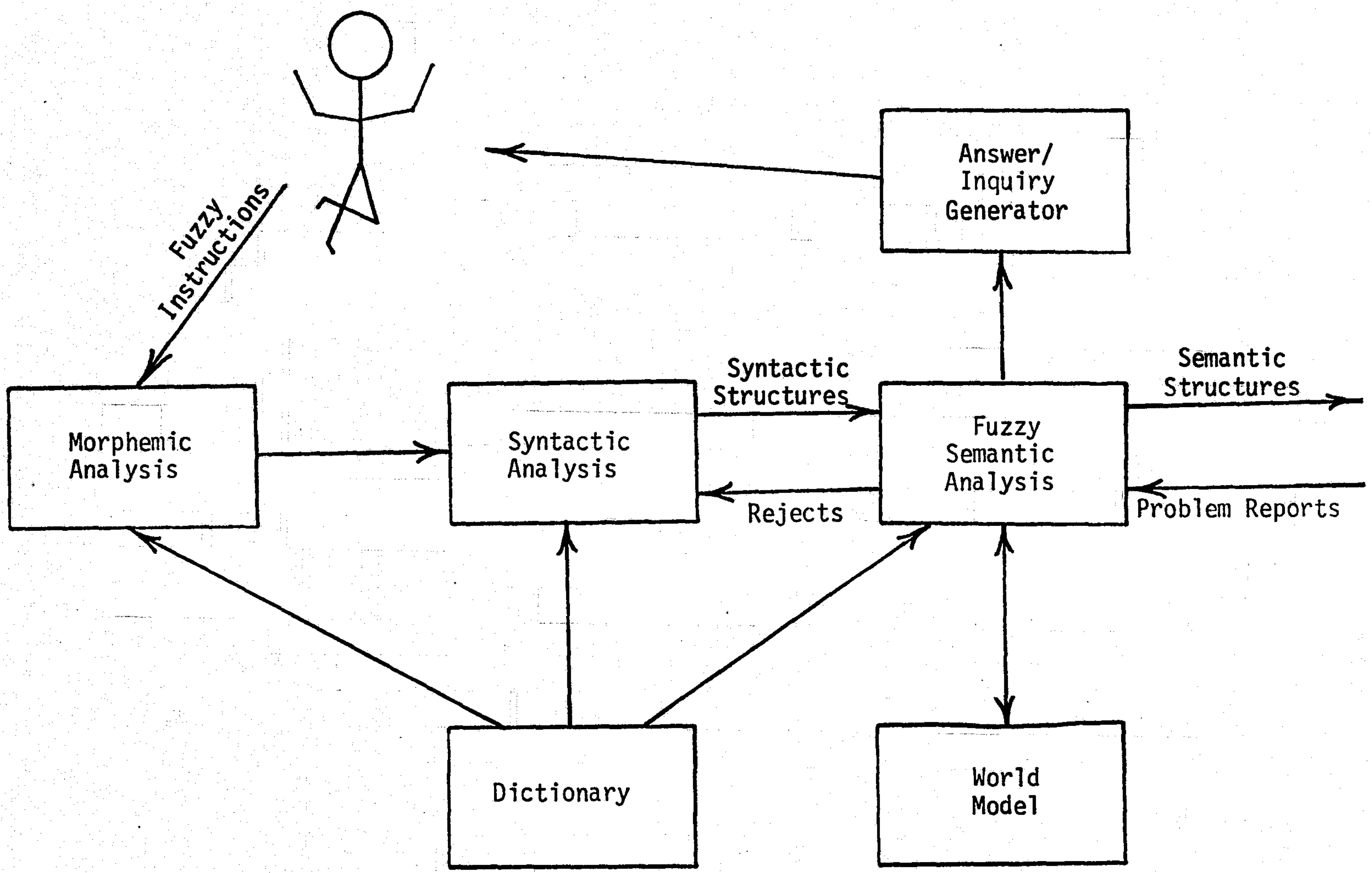


FIGURE 5. FUZZY LANGUAGE TRANSLATOR

3. PLANNING

3.1 The Procedural Net

The planning process is accomplished by building a structure called a "procedural net" (Sacerdoti, 1975) for each high level command given. Each node in the net represents a step in the plan at some level of abstraction. The net development begins at the top with a simple node indicating the operator's command at the highest level of abstraction. This task is then broken down into a sequence of subtasks and is represented as a connected chain of nodes at the next lower level of the net. Each "son" node is connected to its "father" node by an arc in the net. Further, the time sequence of subtask execution is reflected in the links connecting the sons in a linear chain. These time-links are maintained as the net is being developed and always exists at the lowest level of abstraction developed so far. Consequently, if the planning process is halted at any time, a complete plan will exist for accomplishing the original task at some level of abstraction. The planning process is carried on until the bottom nodes in the net all represent primitive commands executable directly by the teleoperator. A procedural net is illustrated schematically in Figure 6.

3.2 The Structure of a Node

Since each node represents a step in the plan at some level of abstraction, it has a goal -- or statement of intention. It can thus stand alone as a task unit. It is connected by double-linked lists to its "relatives": (1) its "father" (the node in the hierarchy of which it is a part), (2) its previous and next "brothers" (empty if it is a first or last son), and (3) its sequence of "sons" (the detailed expansion of its own definition).

A node has additional information that is vital to successful planning (see Figure 7). This information is stored at the node along with the links to adjacent nodes. A node has:

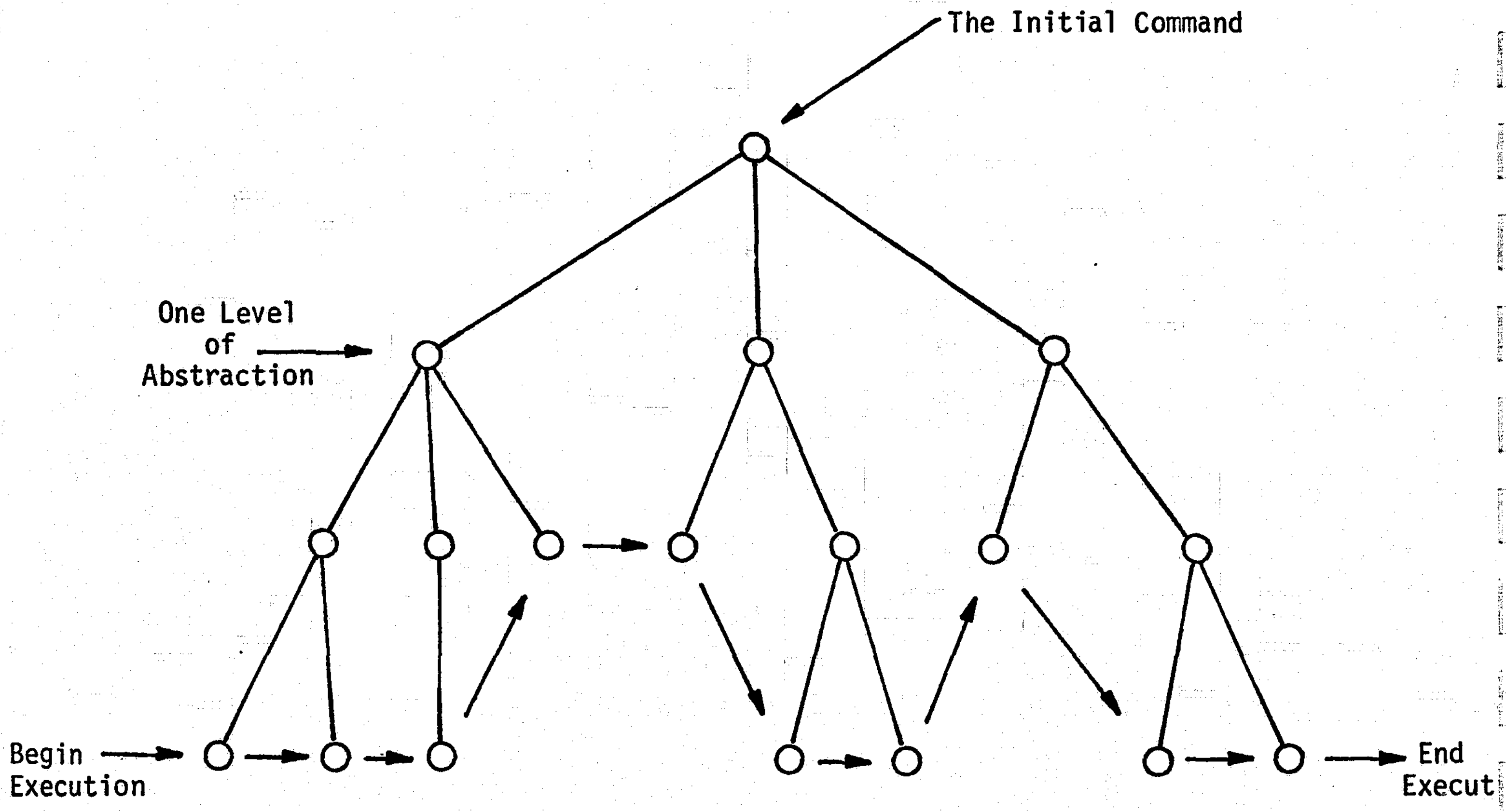


FIGURE 6. A PROCEDURAL NET

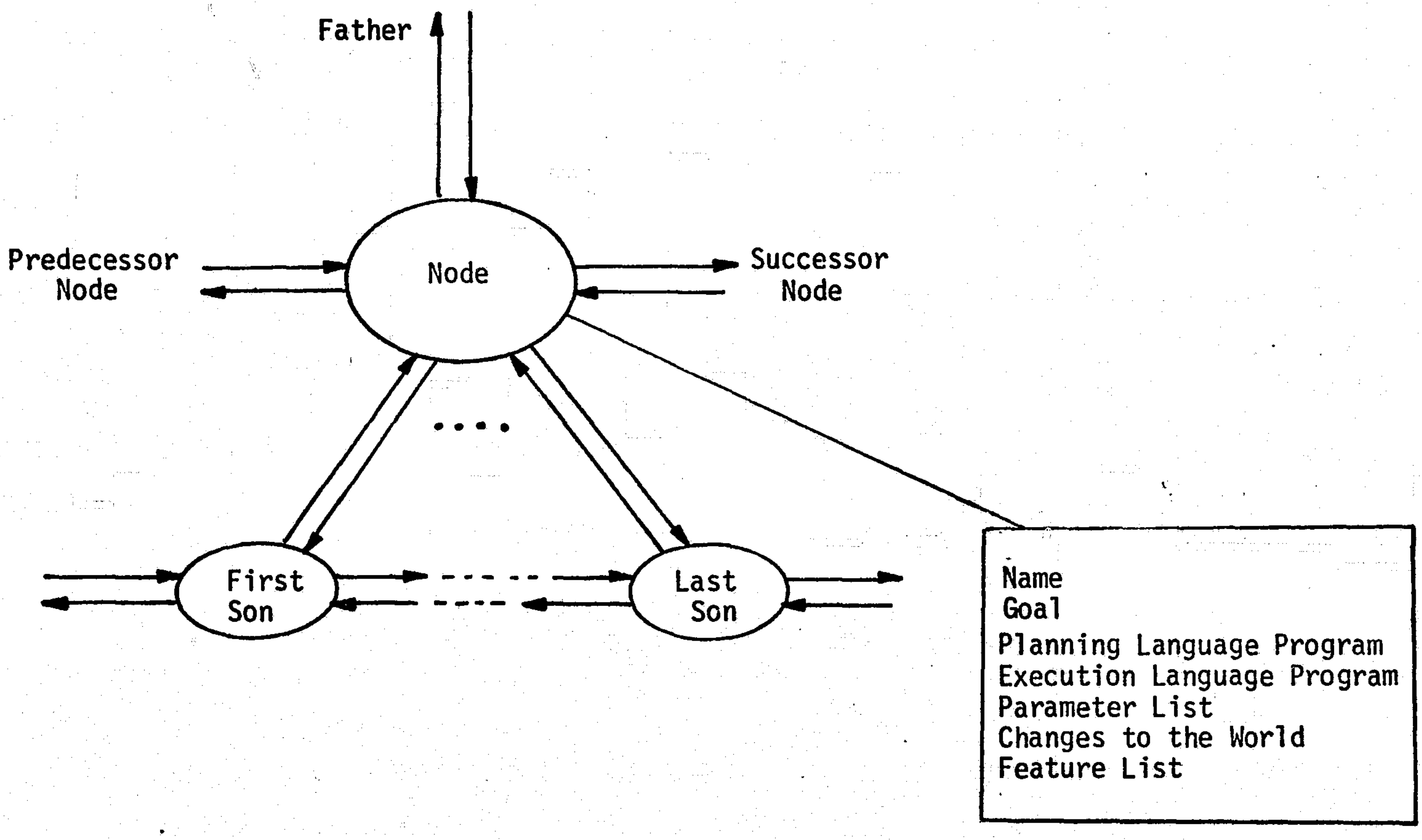


FIGURE 7. DETAILED STRUCTURE OF A NODE

(1) a name which is either specified by the user or implied,

(2) a goal stated in terms of a subtask,

(3) a planning language procedure that is a program for producing the next generation of the node,

(4) an execution language procedure which is executed when the plan is carried out (lowest level only),

(5) a parameter list which is acted upon by the node,

(6) a list of changes to the world model (a procedure that modifies the world model data base), and

(7) a feature list which summarizes information about the particular action the node represents.

3.3 The Planning Language Interpreter Algorithm

The interpreter has a set of ready-made programs which represent the planning semantics of the problem domain. Statements in the language can generate nodes in the procedural net, test conditions in the real world, and compare features of nodes in the net. The interpreter develops each node one level down at each step. That is, it generates all the sons of a node and puts them at the end of an "OPEN" queue to be interpreted at a later time. The basic operator which generates a node is PLAN which has the following syntax:

```
PLAN        NAME (ARGUMENTS)
            GOAL STATEMENT
            SUBPLANS
            EXECUTION EXPRESSION
            WORLD MODEL CHANGES
            FEATURES
END
```

For example, a typical sample plan for a "transfer" command is:

```
PLAN TRANSFER (OBJ1 LOC1 LOC2)
          GOAL (QUOTE TRANSFER OBJECT)
          PLAN (PICKUP (OBJ))
          PLAN (CARRY (LOCATION OBJ (OBJ)) LOC2)
          PLAN (PUTDOWN (OBJ))
          WMSETLOC (OBJ, LOC2) (i.e. world model, set location of
                                     OBJ TO LOC2)
          FEATURES (...)
END
```

When this program (which is associated with the node "transfer") is interpreted, each PLAN operator within the definition establishes a new sibling node (subplan) putting the appropriate call in the program slot. The sequencing time-links are also established at this time. (See Figure 8)

The node expansion algorithm establishes new nodes in a "breadth-first" order. That is, all nodes at one level are created before any nodes at lower levels. Figure 9 shows an example of node expansion using breadth-first ordering. The nodes in the example are numbered in the order of generation. "Expanding" a node means interpreting the plan language expressions associated with the definition of the node and creating all "sons". As each node in a given abstraction level is expanded, all nodes in the succeeding level are created. Then, these are expanded, etc. Finally, the lowest level is reached and the net is complete. The time-links are kept up-to-date during node expansion.

The flowcharts in Figures 10 and 11 describe the detailed algorithm of node expansion. Figure 10 shows the control loop that expands one node at a time. The expansion mechanism requires the maintenance of a "first-in-first-out" node stack in order to monitor the order of expansion. It is initialized with the given command node. During the operation of the algorithm, the node being expanded is found at the top of the stack. This basic algorithm generates a flat tree procedural net with a linearly-linked

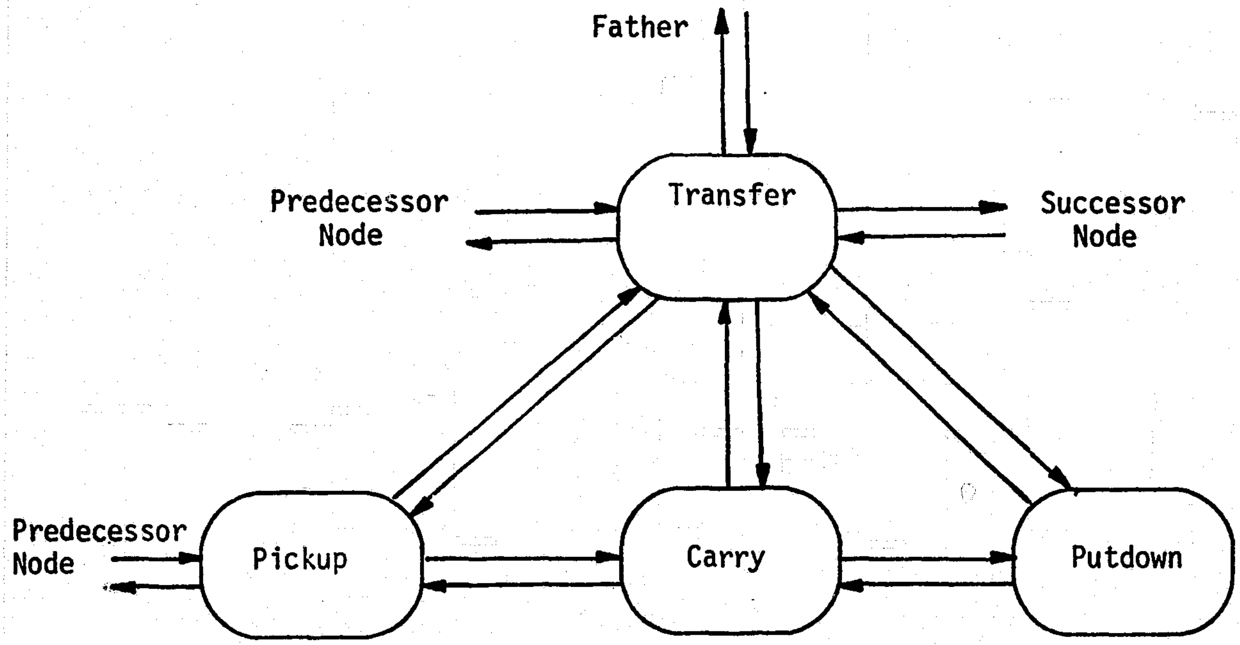


FIGURE 8. THE EXPANSION OF THE NODE "TRANSFER"

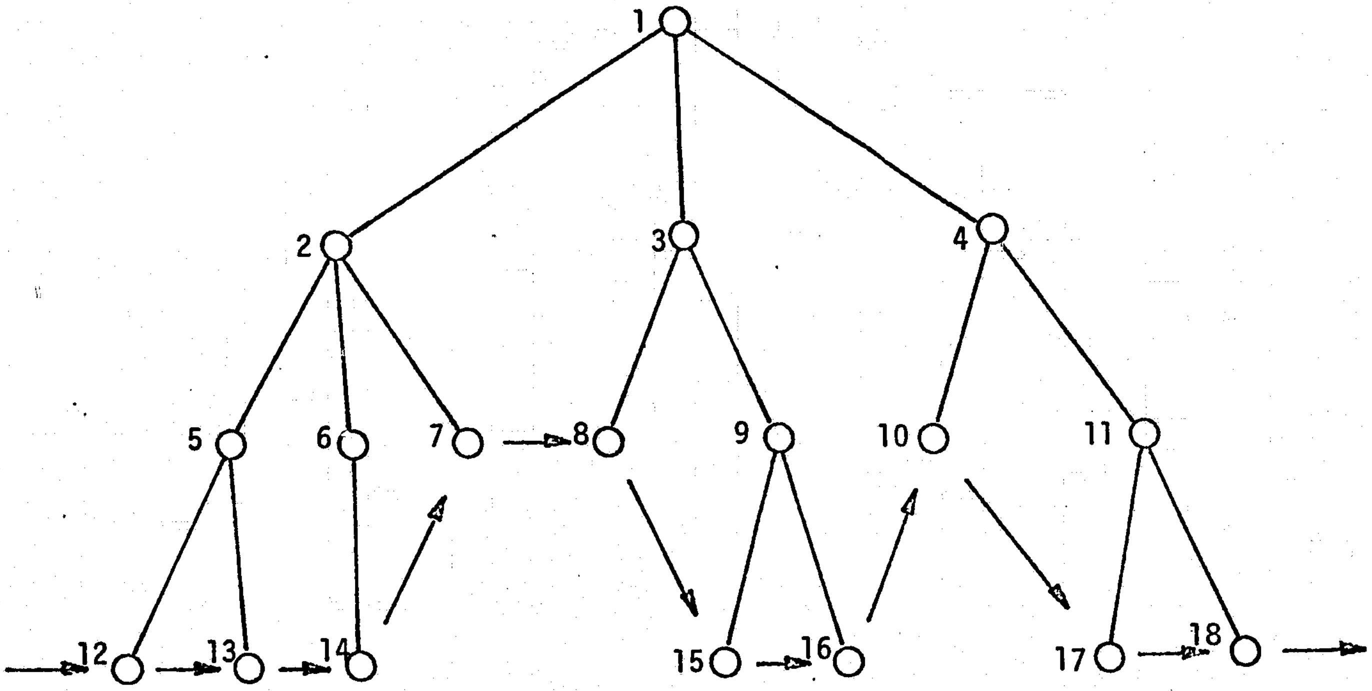


FIGURE 9. BREADTH-FIRST ORDER OF NODE EXPANSION

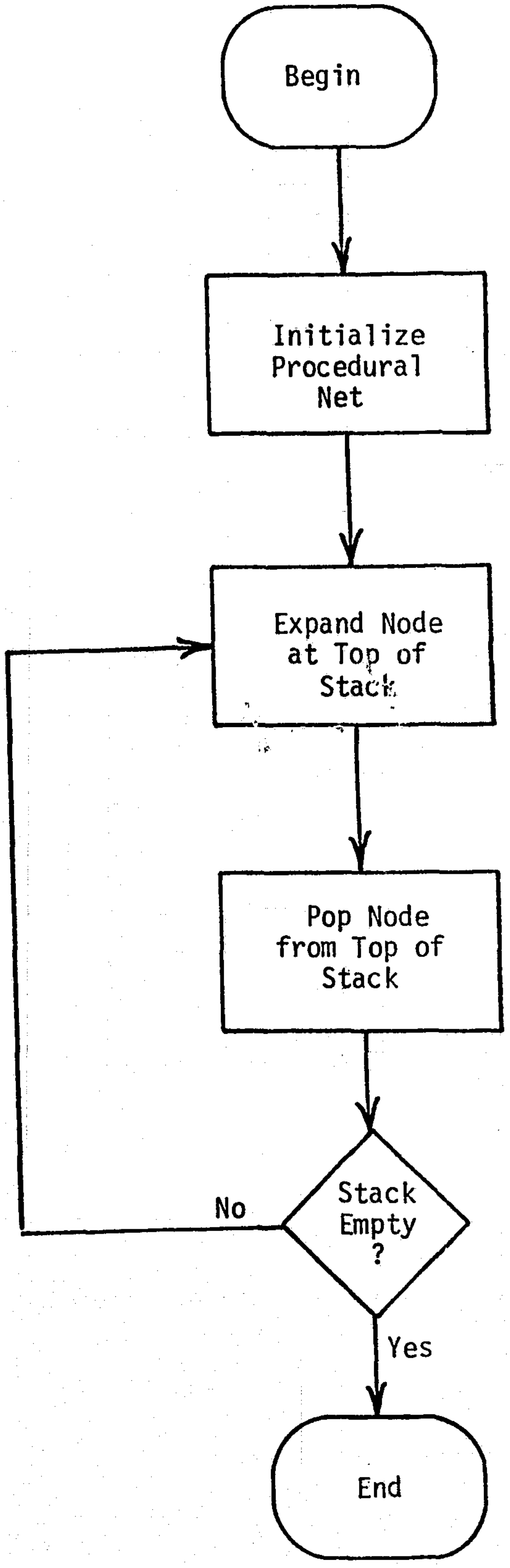

Begin
Initialize Procedural Net
Expand Node at Top of Stack
Pop Node from Top of Stack
Stack Empty ?
No
Yes
End

FIGURE 10. EXPANSION ALGORITHM CONTROL LOOP

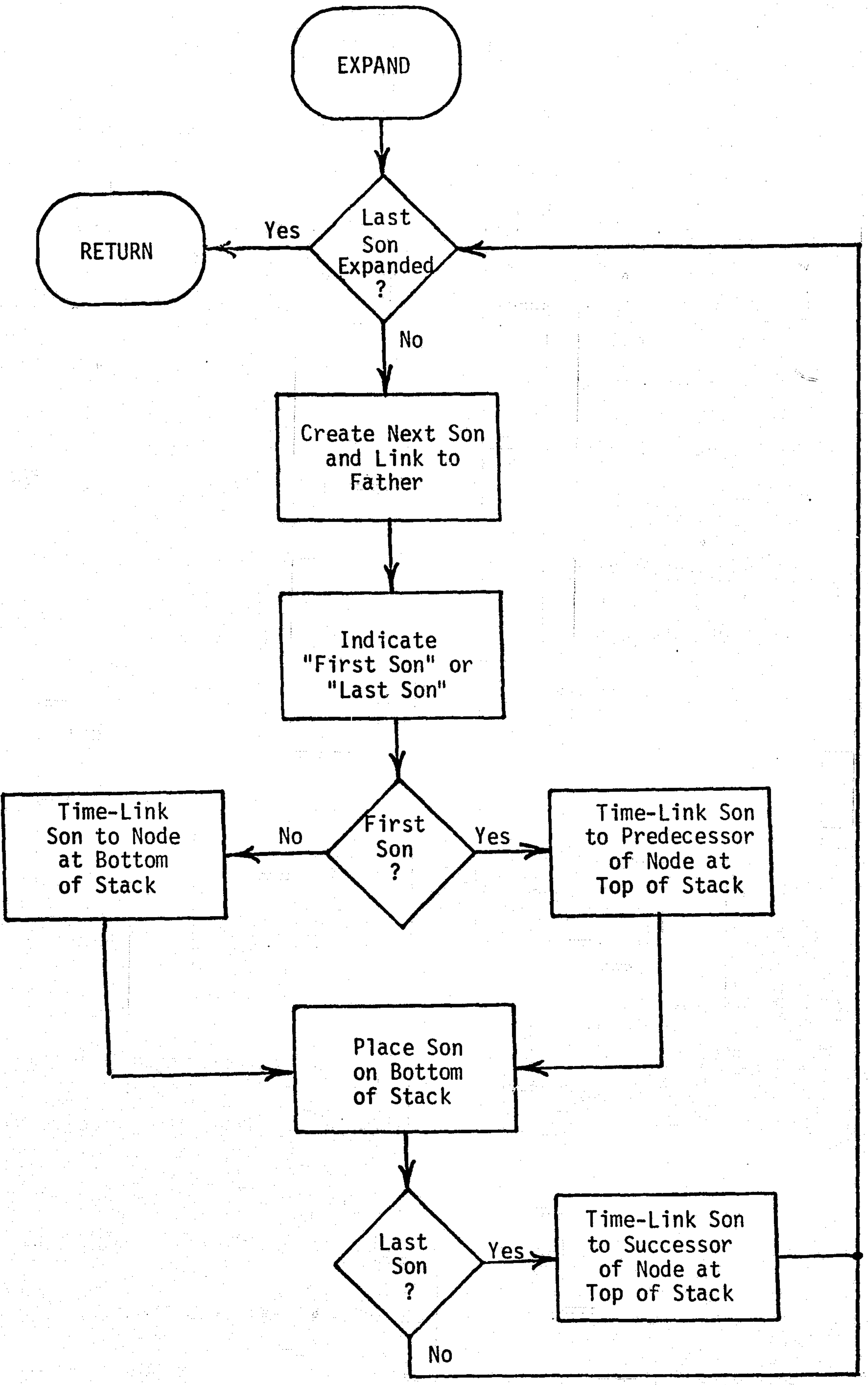

EXPAND
Last Son Expanded ?
Yes
RETURN
No
Create Next Son and Link to Father
Indicate "First Son" or "Last Son"
First Son ?
No
Yes
Time-Link Son to Node at Bottom of Stack
Time-Link Son to Predecessor of Node at Top of Stack
Place Son on Bottom of Stack
Last Son ?
Yes
No
Time-Link Son to Successor of Node at Top of Stack

FIGURE 11. NODE EXPANSION ALGORITHM

execution sequence at the bottom. A detailed example of the algorithm can be found in the Appendix.

3.4 Context Dependency

Developing the procedural net in a breadth-first order, so that each node is expanded to several nodes, is equivalent to a context-free generative grammar where the decision as to which expansion to choose depends on the state of the world. In the proposed system it is possible to add a flexible facility for context dependency. A set of pointers (and operators to move them up and down the net) must be available so that predicates relating to conditions of different nodes in the tree can be evaluated and used for planning. The availability of all of the information in the procedural net and the possibility of conditioning the expansion of a given node on the contents of any node in the net, make the planning procedure more general and powerful. For example, the following expression shows how different nodes can be created depending on a given condition.

```
•
•
•
IF HEAVY (OBJ) THEN PLAN (PUSH (OBJ, LOC1, LOC2))
                ELSE PLAN (TRANSFER (OBJ, LOC1, LOC2))
•
•
•
```

The expansion thus depends on the properties of the object associated with one of the planning variables.

3.5 Backward Planning

An additional dimension of flexibility which can be called "backward planning" can be added to the system if the operations of deletion and insertion into the net are available. Using predicates that evaluate conditions on the net or in the world model, any previously created subtree can be changed based on current knowledge.

Figure 12 shows an example of backward planning. One of the sons of node P is deleted by planning expressions at node S. Typical operations that can be performed include deletion, insertion, interchange of subtrees, copying features from one node to another, etc. This is called backward planning because the active node modifies the structure of nodes already on the net. Since more information is available at a later time, it can be used to modify previously constructed plans.

3.6 Forward Planning

In many cases, planning decisions must be temporarily deferred until information is available at a lower level. This can be accomplished by "forward planning" which allows conditional expressions to be stored at nodes not yet interpreted. Thus, the particular conceptual step associated with a single node need not be determined until the last moment before interpretation. For example,

```
PLAN (IF HEAVY (OBJ) THEN PUSH (OBJ, LOC1, LOC2)
            ELSE TRANSFER (OBJ, LOC1, LOC2))
```

As shown above, the decision to PUSH or TRANSFER is made after one complete additional net level has been constructed thus providing information required for the decision. Notice that, contrary to the expression shown for backward planning, the conditional statement is inside the scope of the PLAN statement. The net structure would appear as shown in Figure 13.

This can be accomplished by storing the expansion expression at the node itself. Later, as the node is about to be expanded, its definition (and name) can be determined based on new information.

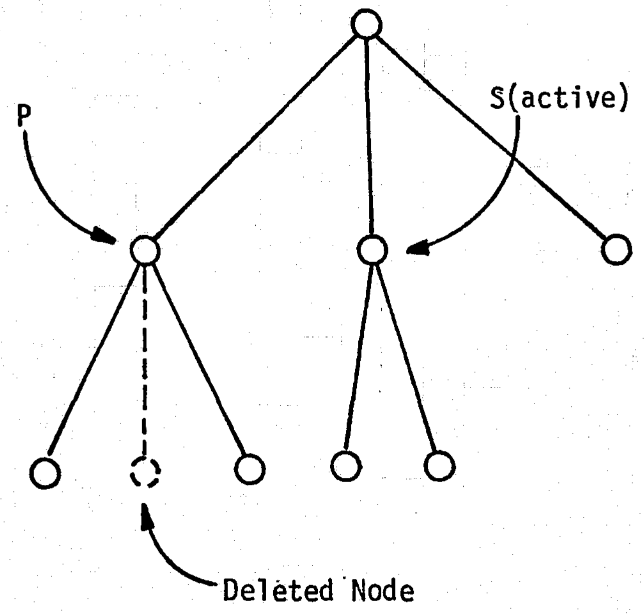
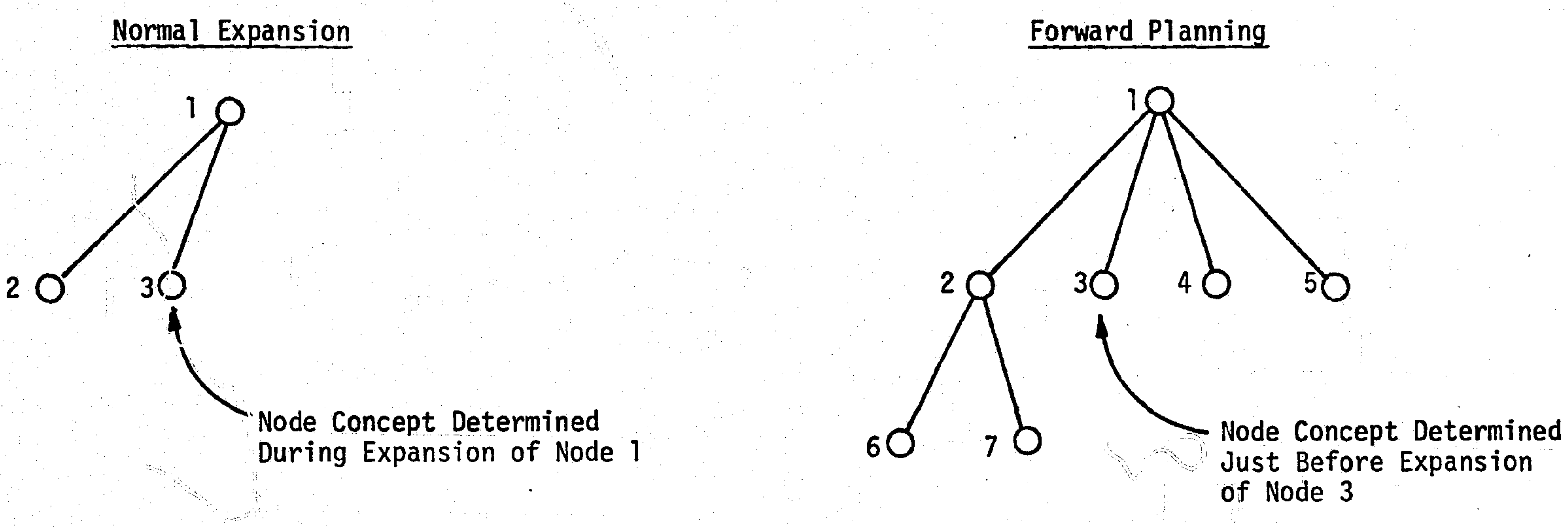


FIGURE 12. BACKWARD PLANNING



Normal Expansion

Node Concept Determined During Expansion of Node 1

Forward Planning

Node Concept Determined Just Before Expansion of Node 3

FIGURE 13. THE NET AT NODE DETERMINATION TIME

3.7 Operator/User Interaction

When the planning is stuck or an action fails to accomplish the "promised" changes to the world, a message is sent to the operator. The hierarchical organization of the net and the declaration of purpose at each node are very helpful in making these messages easy to generate and meaningful to the operator. When a failure occurs, a message is constructed from the path from the failure node in the hierarchy to the top-most node (see Figure 14). Further dialogue can clarify the relations between these nodes and the global plan.

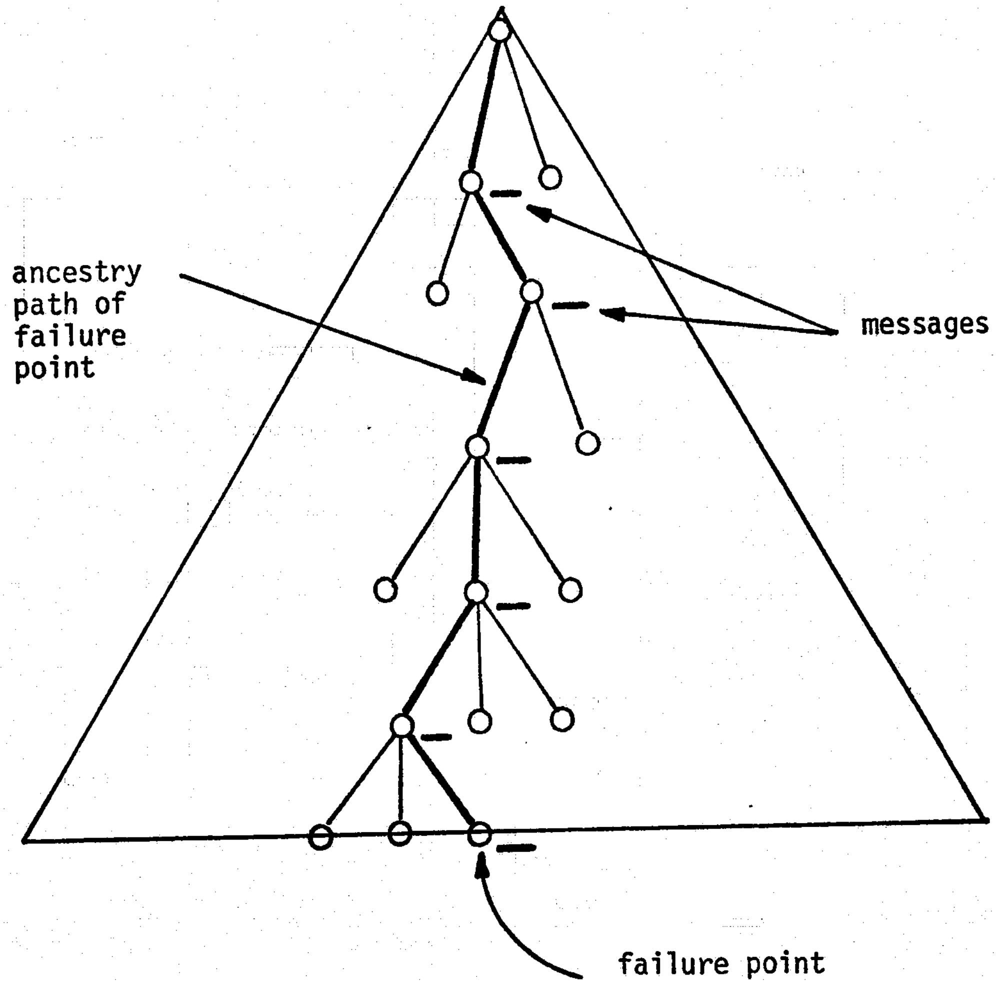


FIGURE 14. FAILURE MESSAGE TRACING

4. EXECUTION

4.1 Execution Control

The planning process -- developing the procedural net -- proceeds as described in the previous section until the net is fully expanded, that is, when all the nodes at the bottom level of the net contain only execution-time commands. In this final form, a continuous chain of time links is threaded from the START indicator through all primitive actions at the bottom of the procedural net in order of their planned execution to the terminating END indicator.

The EXECUTION control interpreter accepts such a fully developed procedural net as input and proceeds, following the time sequencing links, to interpret the execution language expressions associated with each node. There are three types of expressions:

1. Sensor-controlled primitive actions
2. Execution-time planning expressions
3. Execution-time branching expressions.

Each of these expression types serves a special function in the execution sequence.

4.2 Sensor-Controlled Primitive Actions

The basic mechanism by which the system can control manipulator actions using sensory information is the TOTE loop, which stands for Test Operate Test Exit. (See Figure 15) The operation or action to be performed is defined as incremental actions executed from the current state of the manipulator. For example, a CLOSEGRIP command would cause the gripper

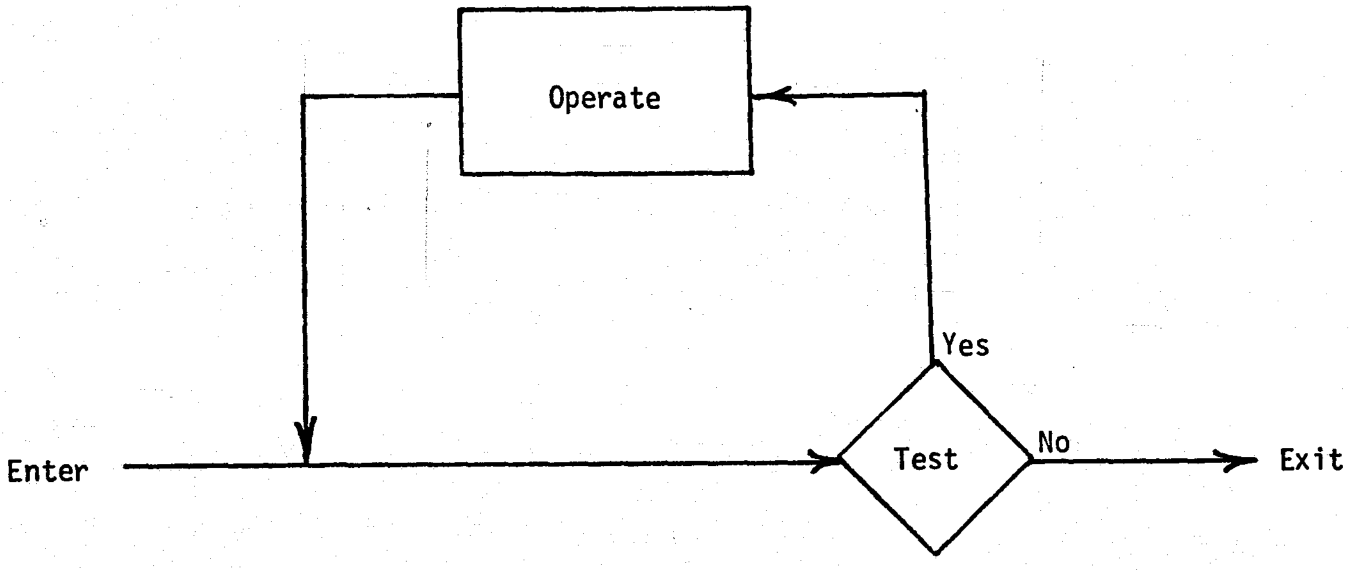


FIGURE 15. THE TOTE LOOP

to be closed some incremental distance from its current position. Similarly, MOVE (X,Y,Z) would cause an incremental move of the manipulator in the direction of the vector X,Y,Z.

The test block represents any one of a variety of conditions that may cause the termination of the execution of a TOTE loop. For any given action there will be a collection of pending conditions controlling its termination, each with a varying scope of influence. Consequently, the conditions are not organized in a simple loop mechanism as shown above. As shown in Figure 3, the block diagram of the Low Level Control subsystem, the Execution Control interpreter accepts procedural net plans as input. It establishes at each step in the plan the appropriate set of condition monitors for this step. It then issues a command to the block which calculates the actual move with the proper set of parameters and initiates the manipulator motion. This motion continues in incremental steps until one or more of the pending conditions generates an interrupt. Execution Control will then exit from the current expression, terminate the motion, and proceed to the next step. Notice that the conditions that are monitored in the "Pending Conditions" block may have several origins:

1. a pattern in the sensory information,

2. a pattern in the manipulator position,

3. a relation or event in the world model which is continuously updated,

4. a time related condition,

5. a combination of these.

The Execution controller repeats these steps for each node in the procedural net time sequence until the END node is encountered.

4.3 Primitive Action Syntax

A condition which controls a given action may cause one of several types of exits from the execution loop. These types are indicated by special key words whose meaning is explained below. The content of an execution node in the procedural net would thus have the form of a multiple exit branching point. The following are typical examples of execution condition control statements.

IF-FAIL *expression*
WHILE *condition* IF-FAIL *expression*
UNTIL *condition* THEN *expression*
DEFAULT-END *condition* THEN *expression*

The various key words cause the following type of exits.

a. UNTIL *condition* THEN *expression*

This is the intended exit from the action. When the condition is satisfied, the manipulator is at the desired position. In the THEN clause, the functions which would make the proper changes in the world model are indicated or possibly some other actions needed before the next manipulator motion is initiated.

b. WHILE *condition* IF-FAIL *expression*

These conditions must hold true throughout the duration of this primitive action. For example, when transferring an object from point A to point B, the object must be sensed between the grippers during the entire motion. If this condition fails before the normal termination condition is satisfied, the IF-FAIL expression will be executed.

c. IF-FAIL *expression*

This branch is taken if the manipulator fails to execute the command given to it, that is, the position monitor did not record the proper change in the links' positions. The expression is defined as the series of actions that will be taken to overcome the problem.

d. DEFAULT-END *condition* THEN *expression*

This branch holds a default test to avoid an infinite loop on the incremental action or possible damage to the manipulator itself. For example, if a grasp action closes the gripper without touching any object (i.e., the UNTIL condition is not satisfied but the WHILE condition is) the action must be terminated and an alternative plan developed.

4.4 Wider Scope Conditions

In the discussion above, all the four types of termination conditions were associated with a single primitive tip node in the procedural net. In general, the hierarchical organization of the procedural net can be utilized to allow execution-time conditions to be associated with <u>any node</u> in the hierarchy. The monitor controlling the condition associated with such a high-level node would survive during the execution of all its descendents. That is, the high level node would have a "scope" of operation. Each node in the net which is a first descendent or a last descendent already has an indicator to that effect (see net expansion algorithm). When the execution controller enters a tip node that is a first descendent, it will climb the "father links", establish all the conditional monitors it encounters until it reaches the first node which is not a first son. This is shown in Figure 16. Similarly, when a last descendent node is encountered, a climb on the "father links" will reveal which condition monitors should be deactivated.

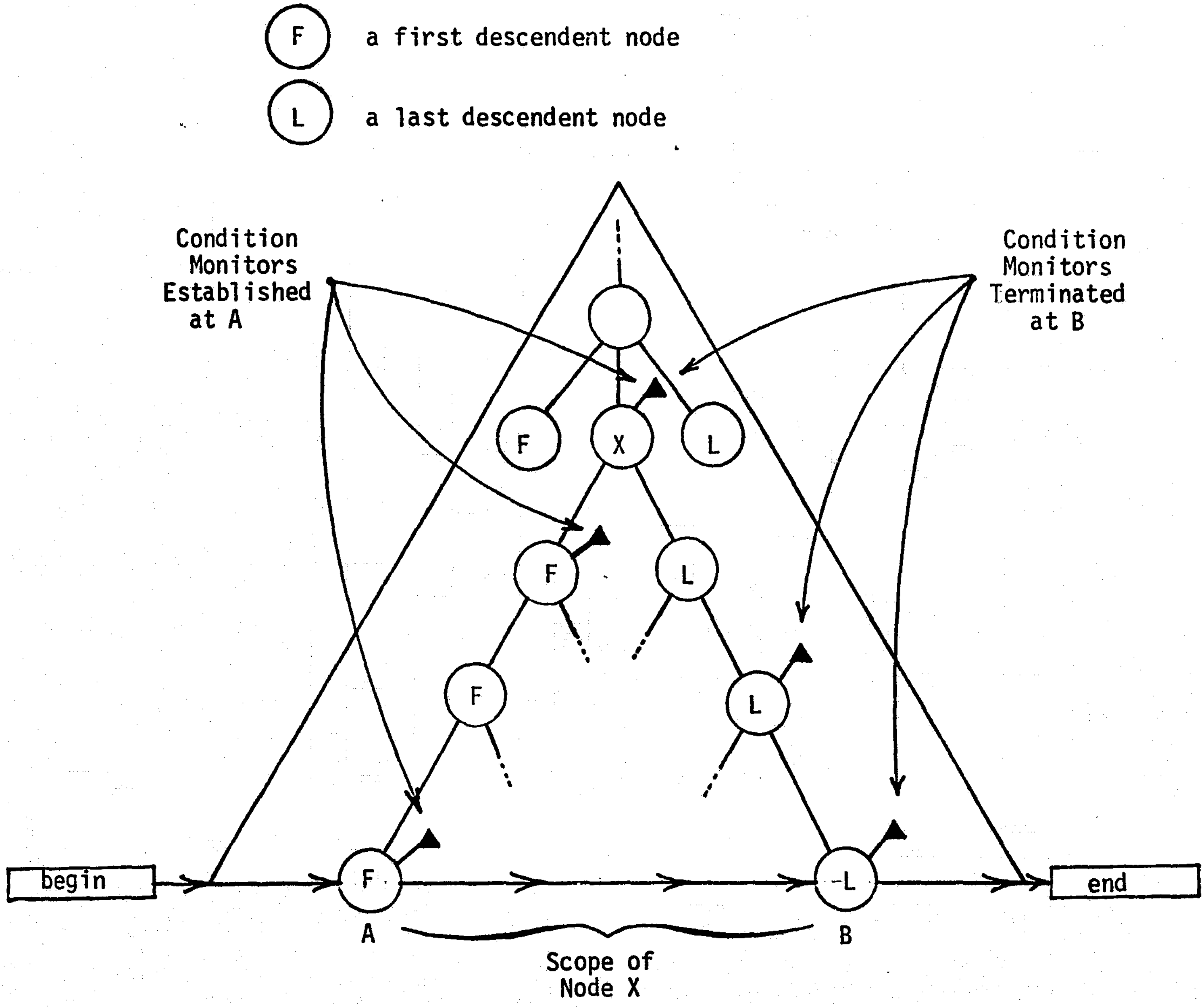


FIGURE 16. CONDITION MONITORS

Extending this capability to the limit, conditions associated with the top node in the net and their corresponding monitors are alive globally and are always there to guard against global damages. For example, such a monitor may watch for any motion which reaches the limit of the manipulator's extension capabilities.

4.5 Transfer of Control Between Planning and Execution

The relation between planning and execution is not a simple one-way transfer of control. On one hand, it may happen that some information needed for planning is not currently available and can be obtained only by plan-time actions. On the other hand, it is not possible to plan ahead for all eventualities either because some of the possibilities are very rare and the planning effort is not justified, or because not enough information is available at planning time and details of the situation are needed to plan that specific action. In such cases, it is possible to resort both to planning during execution and to execution during planning.

4.5.1 Plan-Time Execution. If an item of information is needed that cannot be found in either the procedural net or the world model data base, planning can be suspended in order to initiate execution of a reconnaissance task. This capability allows a recursive activation of the entire system which constitutes planning, action execution, and information storage. (See Figure 17) When control is returned to the planning phase of the original task, the required information should be in the world model data base.

4.5.2 Execution Time Planning. In some cases, it is necessary to defer planning to the execution phase. It may happen that information available only at execution time is required to generate a plan, or that sequence of actions is such a rare case that the effort of expanding a plan for it is not justified at the initial planning phase. For example, during

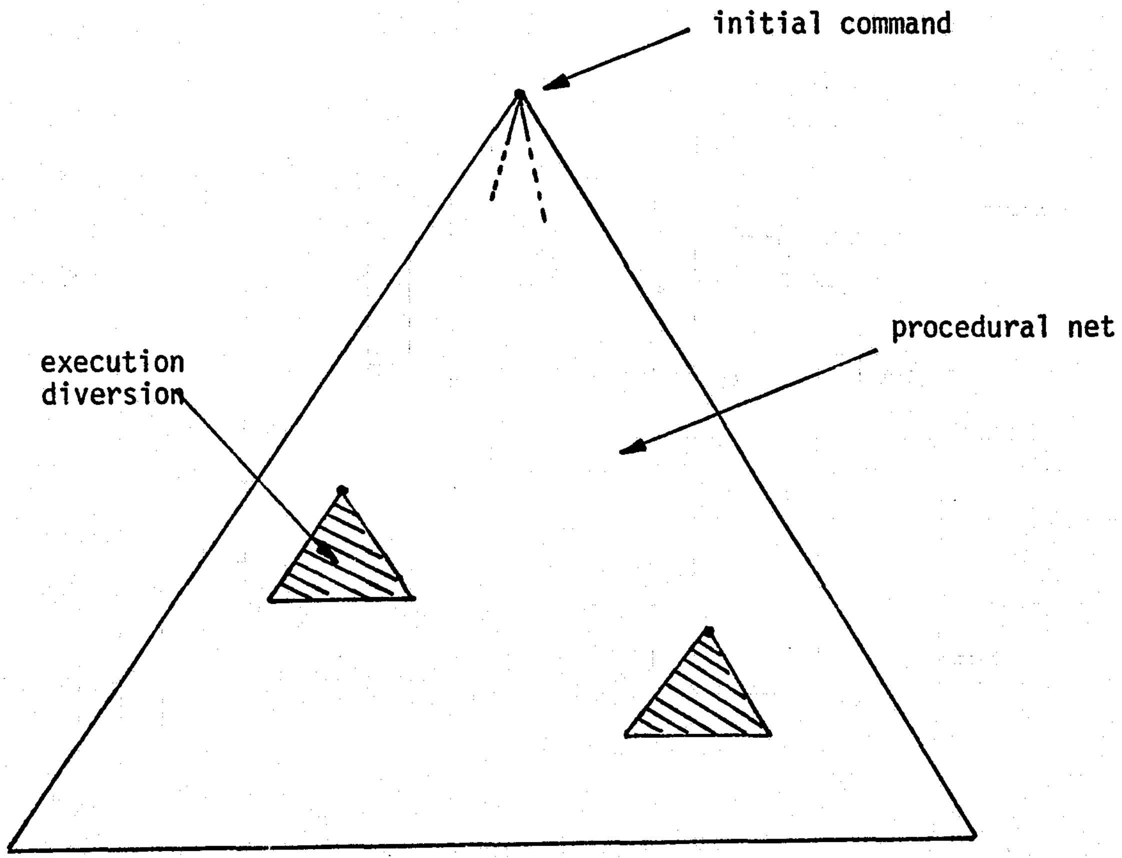


FIGURE 17. PLANNING-TIME EXECUTION

a task of carrying an object, a WHILE condition may test if the carried object is still between the grippers. If it fails, planning is initiated for a PICKUP.

```
MOVE (OBJ)
   :
   WHILE HOLDING IF-FAIL PLAN PICKUP
   :
```

The PICKUP plan will use information available in the current state of the world and at the time the object was dropped. An expression at a tip node may actually call the entire planner system into action. The manipulative action will be halted during this planning phase and resumed afterwards. Figure 18 shows this process schematically.

4.6 Execution Time Branching

In some cases, decisions cannot be made at planning time because the choice of action to be taken depends on the outcome of an execution-time test. For example, in the plan to pick up a tool, like a screwdriver, a different approach trajectory must be taken depending on whether the tool is lying flat on a surface, or is standing vertically in a tool box. This fact is not always known in advance.

Adding this capability of deferring decisions to execution time enhances the flexibility of the planning system. Corresponding changes would be made to the procedural net and its expansion algorithm. For example, in the procedural net described in Section 3, each node had one sequence of time-linked sons. With the execution-time branching option, a node may have several alternative sequences of descendents. The proper alternative sequence is chosen during execution. The procedural net can be visualized as having a "flat bottom" rather than a single thread of time links.

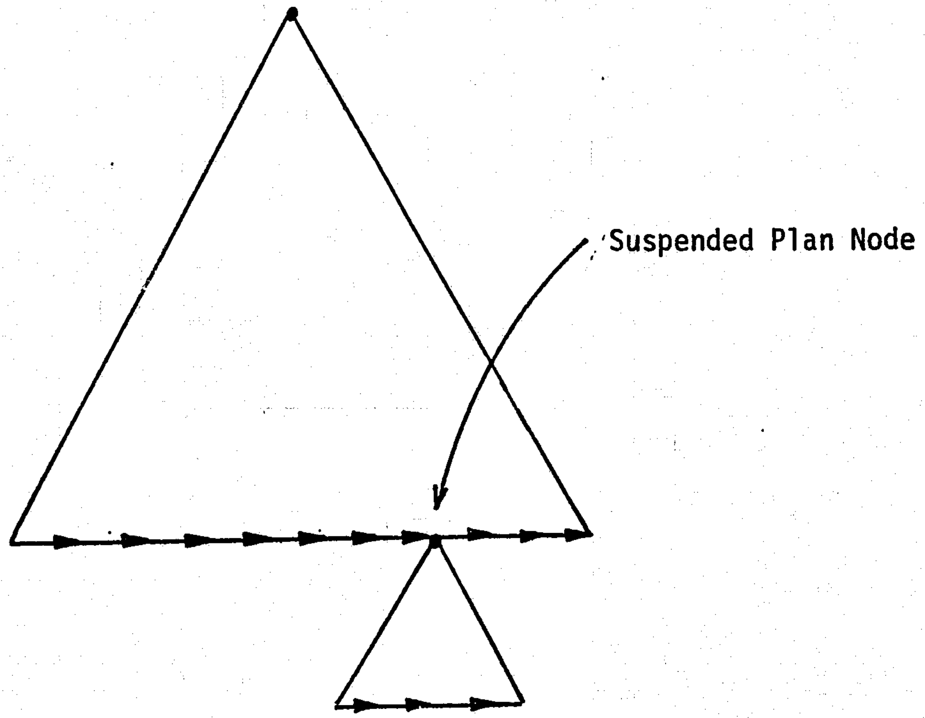


FIGURE 18. EXECUTION-TIME PLANNING

4.7 A Sample Scenario

The following is an example of the sequence of events that will take place when the autonomous control system accepts a command such as:

Put Rock #12 on the shelf.

This is a high-level task-oriented instruction. The language translator will first transform it into a semantic structure. This structure will be input to the planner as the top node of the procedural net. The planner will develop a detailed plan to execute the command by expanding the procedural net as appropriate for the current state of the world. The completed procedural net is then given to the execution controller which follows the bottom links in the net, using sensory information to decide on substep terminations. When the plan ends, a completion message is sent to the operator..

If any problem arises during planning or execution, the system can communicate to the operator, by extracting information from the procedural net, the exact place in the plan the problem arose. It can also indicate what condition failed to be fulfilled thus aiding in solving the problem. The corrective command given by the operator replaces the problematic part of the plan and the process can resume as above. In most cases, however, the system will contain enough specific knowledge to overcome small problems without the operator's help.

5. WORLD MODEL

5.1 Typical Tasks

The world model, updated and maintained by the system itself, is largely determined by the type of tasks that the system is expected to perform. The scope of tasks addressed here may be called "pick up and transfer tasks". In this general class we have the following examples:

Transfer objects from one location to another.
Put one object on top of another.
Pour liquid from a small container.
Insert a stick into a hole.
Grasp and lift objects of various shapes.

In performing such tasks, the control language must be capable of specifying (1) restrictions on trajectories, (2) approach paths to a peculiarly shaped object, and (3) constraints on the orientation of the transferred object. The world model must include enough information to plan the appropriate trajectories that avoid known obstacles and to modify the trajectory if an unexpected obstacle is sensed. As an example, a simple world model is adopted which includes a description of the space that can be reached by the manipulator and a representation of objects with the information relevant for the tasks intended. The planning mechanism would be able to extract specific items of data from this structured world model.

5.2 Space

In planning the trajectories in the working space, answers are required for the following types of questions:

Is there an object at location (X,Y,Z)?
Which object is at (X,Y,Z)?
Can the manipulator reach (X,Y,Z)?
What is a safe trajectory above all objects?

To answer these and other such questions, the working space is divided into a convenient tesselation to supply operators with information for retrieval, deletion, and insertion (see Figure 19). The following functions are typical of those necessary to interact with the world model data base.

predicates:	IS-OBJ-AT (Location)	Is object at a given location?
	ARM-REACH (Location)	Can the arm reach a given location?
retrieval functions:	WHICH-OBJ (Location)	Which object is at a given location?
	SAFE-TRAJEC (L1,L2)	Safe trajectory from L1 to L2?
change:	SET-EMPTY (Location)	Set given location empty.
	SET-OCCUP (Location,obj)	Set given location with object.

5.3 Objects

The information retained about objects pertain to the typical tasks: object pickup and transfer. Thus, there is no need in the first approximation for a comprehensive object description. It should be sufficient to store only the dimensions of the minimal box which can enclose the object, even if the object itself is irregular (See Figure 20). The features associated with each object in the world model and the corresponding retrieval operators are described in the following sections.

5.3.1 Shape. Defined as the dimensions of the enclosing box in order of decreasing lengths: length, width, depth. Operations that retrieve such information are:

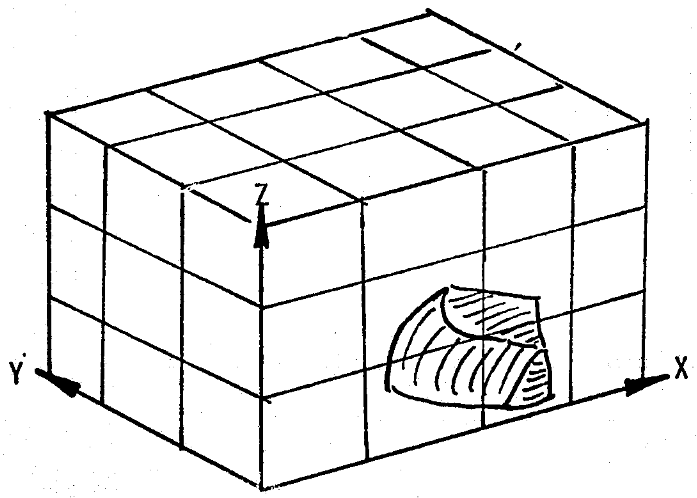


FIGURE 19. THE SPACE TESSELATION

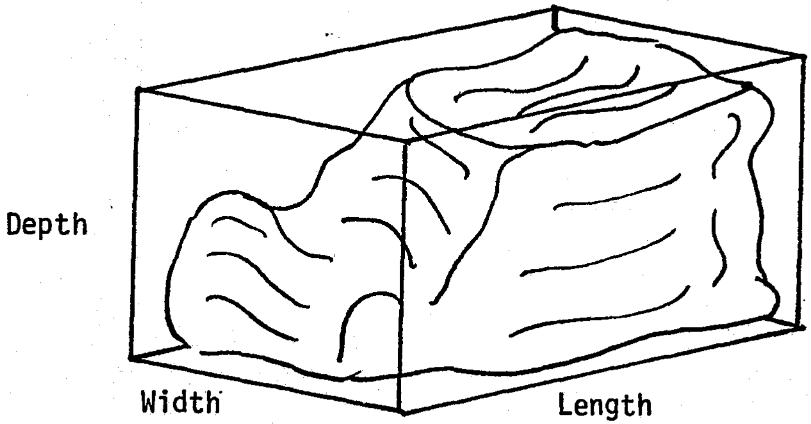


FIGURE 20. A MINIMALLY ENCLOSED OBJECT

LENGTH (obj)
WIDTH (obj)
DEPTH (obj)
SHAPE (obj)

5.3.2 Location. The position of the center of the enclosing box in the coordinate system.

LOCATION (obj)

It may also be necessary to know the space (i.e. tessellation cubes) that an object occupies so that his neighbors may be identified.

OCCUPY (obj)

5.3.3 Orientation. The orientation of the axis of the enclosing box relative to the coordinate system. This information is needed to plan an approach path for grasping the object.

ORIENTATION (obj)

5.3.4 Physical Features. Various other physical features are useful in planning transfer of objects:

WEIGHT (obj)	- to decide whether to lift or push the object
BALANCE (obj)	- to identify problems of weight distribution
SURFACE (obj)	- smooth, rough, irregular, etc.
PLIABLE (obj)	- hard-to-soft in several degrees.
BREAKABLE (obj)	- breakable - to exercise special care
ELASTIC (obj)	- elastic-to-plastic

5.4 Changes to the World Model

When planning a sequence of actions that change some aspects of the real world, specific changes must be taken into account even during the construction of subsequent steps of the plan. Since most of the world model is not changed during any of the actions that take place, it is quite wasteful to duplicate the whole model after every action. The approach taken by Sacerdoti (1975) is adopted which keeps a simple world model visible at all steps (nodes) and only masks aspects which are changed. Thus, the system will retrieve the latest values assigned to those particular facts.

Figure 21 shows how facts about the world are accessed from the current world model. The small blocks represent individual changes to particular facts and are "seen" from the world model at time t_c. Every change "masks" the previous value of that fact. From the current world model (t_c) the latest values of the facts are always available.

At planning time, changes are <u>hypothetical</u>; at execution time, changes are <u>actual</u>. The hypothetical changes are <u>assumed</u> to have been accomplished when developing nodes further in the plan and are used to specify the later states of the world. During action-time, these hypothetical changes can be used as tests for the success of a step. The state of the world is compared with the hypothetical one when the execution control finishes a particular node. When they are similar the action goes to the next node, and if not, a corrective action can be taken.

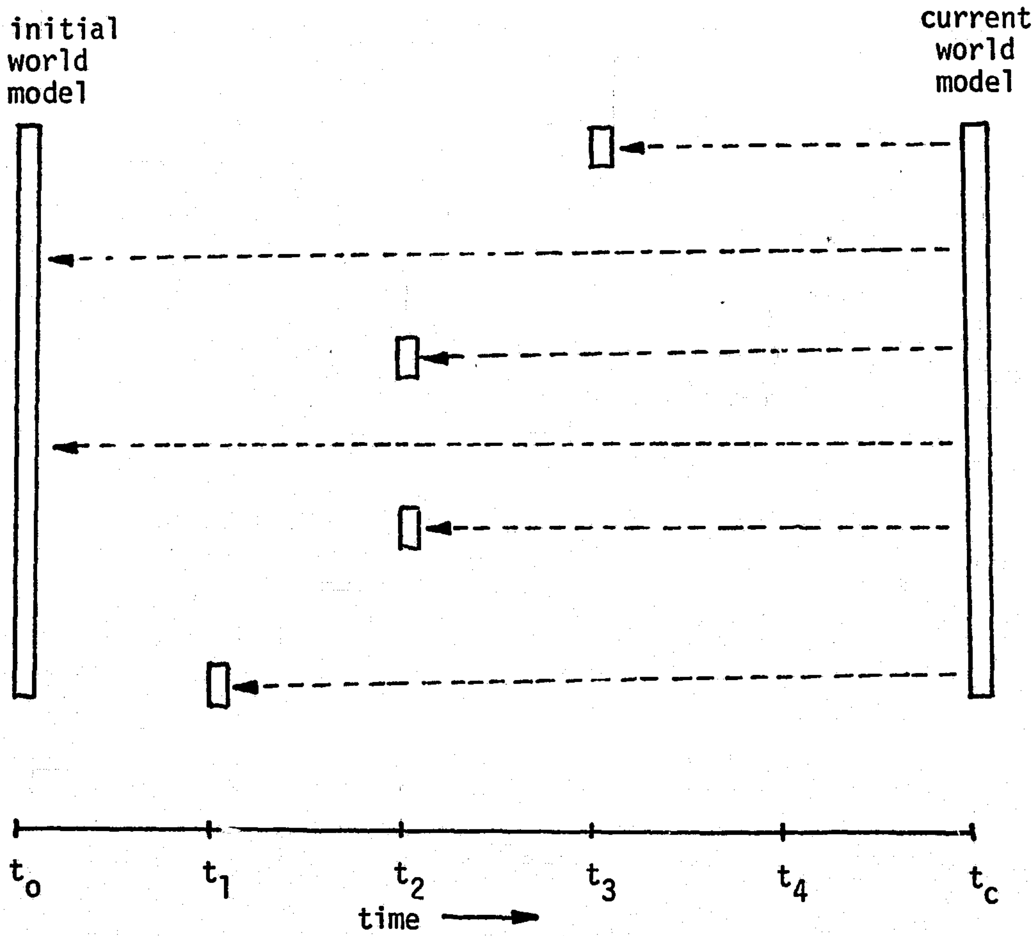


FIGURE 21. UPDATING THE WORLD MODEL

6. ROBUST MANIPULATOR COMMAND LANGUAGE

The ease and efficiency of communication between the human user and a manipulator control system can be enhanced if the commands are given in a robust command language. "Robustness" is the capacity of a system to accomplish its intended task when its input is inaccurate, unreliable, or incomplete. A control system demonstrates robustness if it is able, without external program modification, to adjust to vague instructions or to a slightly perturbed environment. Such a system will usually exhibit graceful degradation of performance as the actual situation encountered gets more dissimilar to the expected input.

The technical approach to achieve this behavior is based on the theory of fuzzy sets (Zadeh 1965; Kaufman 1975) which can be used as a mathematical theory for modeling vague concepts as are commonly found in natural language. Fuzzy sets were advantageously applied to robot planning (Goguen 1974), to language conceptual sets (Zadeh 1965), to algorithms (Zadeh 1968), and many other aspects of concept formation, manipulation, and utilization. Recently, Shaket (1976) used fuzzy sets to model the semantics of object classes and relations of a limited English-like language in a manner which is convenient for reference. The system accepts instructions in the form of simple indicative sentences and responds by indicating the intended reference object. The system demonstrates robustness in the sense that small changes in the instruction or in the state of the world do not disturb it. A sample command in a robust manipulator command language would be:

> PUT THE BIG BAR AT THE CENTER OF THE VEHICLE.
> vb/noun group prep./group indicating location

The term BAR is not a label of a particular object but refers to a fuzzy class of objects. In a particular setting, however, the interpreting system would be able to resolve the expression "THE BIG BAR" into a reference to a specific object in the scene. Notice that such a reference mechanism does not require that the user know precisely size, shape or location of the intended objects. Furthermore, such robust reference would be functioning properly even if the scene is slightly changed. There may be several additional small objects nearby. Additionally, a semantically oriented planning system would know that in order to put an object somewhere, that object must be picked up first, and would add the appropriate steps into the plan it generates. Such a robust command language facilitates the man-machine interaction in several major ways:

a. The user is relieved from the burden of remembering the specific label or exact features of every object in the scene. He can make the reference in a flexible and efficient natural-language-like manner, which will be resolved by the system to a specific reference.

b. The system is organized around semantic programs rather than being syntactically oriented, and would not balk at small syntactic errors. This may be called "syntactic robustness".

c. The system embodies considerable semantic information of the problem domain. Thus it can supply different items of information that are required to perform a command successfully, but were not specifically given in it.

7. SUMMARY AND FUTURE DEVELOPMENTS

An overall design and basic algorithms have been described for an autonomous control system for remote manipulators which utilize sensory feedback. The important points in the system's organization are:

1. Hierarchical, recursive organization of specific knowledge about planning and execution in a particular problem domain. This allows easy definition and use of task-oriented constructs rather than dealing with detailed chains of primitive manipulator operations.

2. Plan Structures, in the form of procedural nets, are explicitly available for analysis and manipulation throughout both the planning and execution phases. This contrasts with other systems where the planning information is hidden unaccessibly in the control structure of the planning algorithm.

3. The procedural net is a combination of data and procedures in a special language. This language has primitives useful in planning and the flexibility of an interpreted execution. The procedures that develop the net (construct, modify, or delete subplans) can guide themselves according to four kinds of data:

   a. general information about the problem domain,
   b. specific information in the updated world model,
   c. planning information contained in the procedural net, and
   d. missing information obtained by plan-time execution of surveillance motions.

4. The procedural net is interpretively expanded in a breadth first order. At every point in the expansion, the whole plan at every higher level of detail is available for interrogation. Thus, both backward and forward planning become possible.

5. The procedural net allows parallel processes to proceed concurrently and to defer decisions until execution time. A specific path of action will be decided according to the results of tests performed on the real world when the plan is carried out.

6. Plan execution is done interpretively by following the time sequencing links in the procedural net.

7. Each primitive action is incremental and is controlled by sensory information in a TOTE loop (Test-Operate-Test-Exit). The tests are continuously monitored by a pool of conditions which evaluate both sensory information and a dynamically modified world model.

8. When problems or rare conditions occur, planning can be initiated during execution to develop a correcting plan. After this diversive plan is executed, the original task can be resumed.

The conceptual design presented in this report is a first step toward a compact working system. Three phases for further development are suggested:

Phase 1. Theoretical development and high-level language implementation of the planning language, execution language, and world representation. Demonstration of performance on a simulated manipulator.

Requirements:

1 man/year
Large computer (PDP 10)
One of the recent AI languages (QLISP)
100 K+ memory

Phase 2. Problem specific development. Programming problem-specific knowledge as related to a specific manipulator in a given set of tasks. Testing autonomous operation in a real environment.

Requirements:

2 man/years
minicomputer with real time interfaces
64 K memory

Phase 3. Implementation. Compression of computer programs to space-worthiness.

Requirements:

1 man/year
minicomputer and manipulator
32 K memory

8. REFERENCES

Ernst, G.W. and Newell, A. GPS: A Case Study in Generality and Problem Solving. Academic Press, 1969.

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

Fikes, R.E., et al. Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 1972, 3:251-288.

Finkel, R., et al. AL, A Programming System for Automation. Stanford Artificial Intelligence Project, Memo No. 243, November 1974.

Goguen, J.A., Zadeh, L.A., et al, ed. "On Fuzzy Robot Planning", *Fuzzy Sets and their Applications to Cognitive and Decision Processes*, Academic Press Inc., New York, 1975, pp. 429-447.

Hendrix, G.G. Modeling Simultaneous Actions and Continuous Processes. *Artificial Intelligence*, 1973, 4:145-180.

Kaufman, A. Introduction to the Theory of Fuzzy Subsets. Academic Press (New York) 1975.

Kuipers, B.J. A Frame for Frames. *Representation and Understanding, Studies in Cognitive Science*, D.G. Bobrow and A. Collins (Eds.), Academic Press, 1975.

Miller, G.A., Galanter, E. and Pribarm, K.H. Plans and Structures of Behavior. Holt, Rinehard and Winston, Inc. (New York) 1960.

Nilsson, N. Problem-Solving Methods in Artificial Intelligence. McGraw-Hill, 1971.

Sacerdoti, E.D. A Structure for Plans and Behavior. SRI Artificial Intelligence Center, Technical Note 109, 1975.

Sacerdoti, E.D. Planning in a Hierarchy of Abstraction Species. Third International Joint Conference on AI, 1973, pp 412-422.

Shaket, E. Fuzzy Semantics for a Natural-Like Language Defined Over a World of Blocks. University of California at Los Angeles, Artificial Intelligence Memo No. 4, 1976.

Shirai, Y. Edge Finding, Segmentation of Edges and Recognition of Complex Objects. Fourth International Joint Conference on Artificial Intelligence, September 1975.

Winograd, T. Understanding Natural Language. Academic Press (New York) 1972.

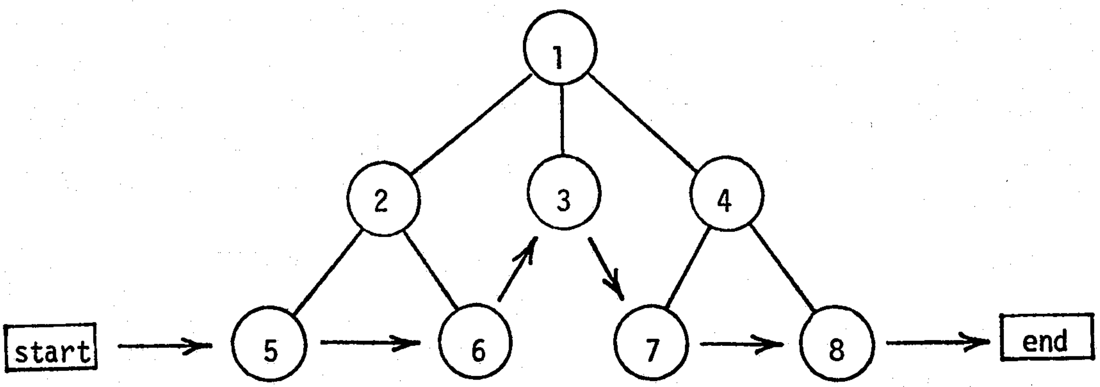
Zadeh, L.A. Fuzzy Sets. Information and Control, 1965, 8:338-353.

Zadeh, L.A. Fuzzy Algorithms. Information and Control, February 1968, 12:94-102.

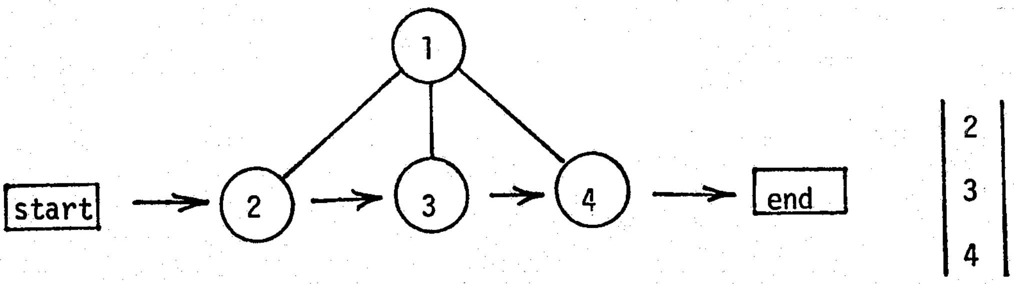
Zucker, S.W. Region Growing: Childhood and Adolesence. Univ. of Maryland, Comp. Science Technical Report TR-370, April 1975.

APPENDIX: PROCEDURAL NET EXPANSION EXAMPLE

The following example will illustrate the detailed operation of the node expansion algorithm as shown in Figures 10 and 11. Assume that the user has given a high-level command which, when expanded completely, will decompose into the following procedural net representation.



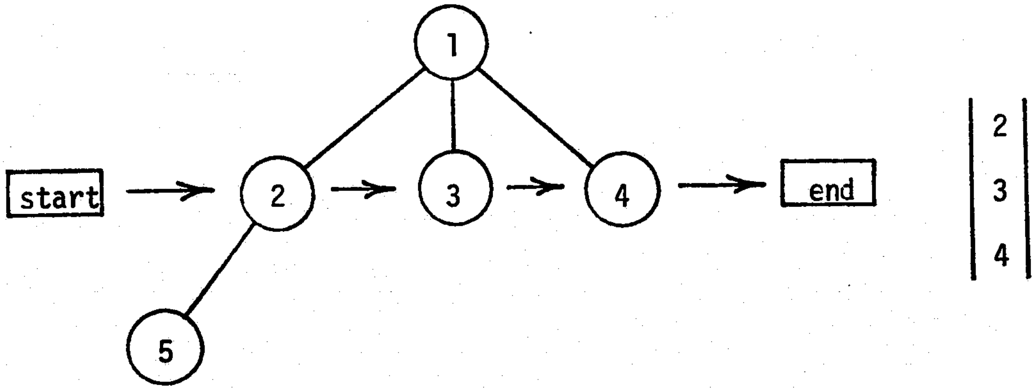
Assume further that the algorithm has already expanded the initial command (node 1) for one complete layer. The current net, therefore, appears as follows along with the contents of the node stack.



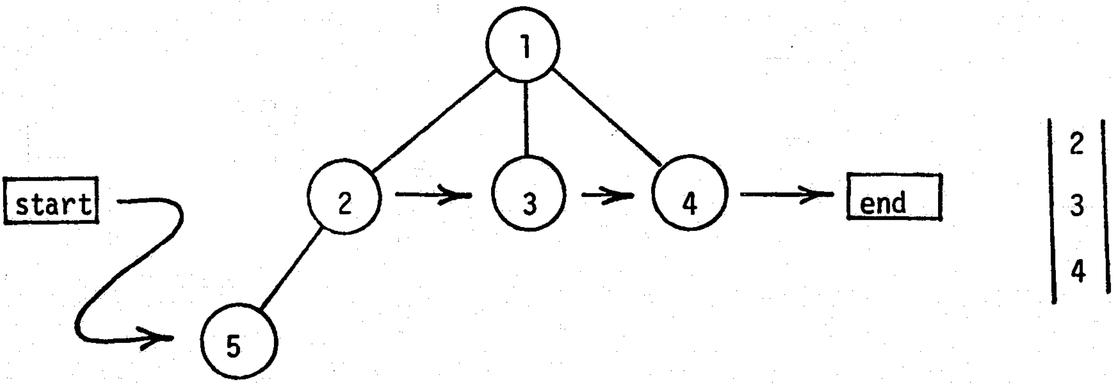
Since node 1 has just been completely expanded and popped from the top of the stack, the algorithm now calls for the expansion of node 2 (see Figure 10). The steps of the algorithm will be followed showing the progress toward completion.

1. Expand node 2 (top of stack).

2. Create first son (node 5).

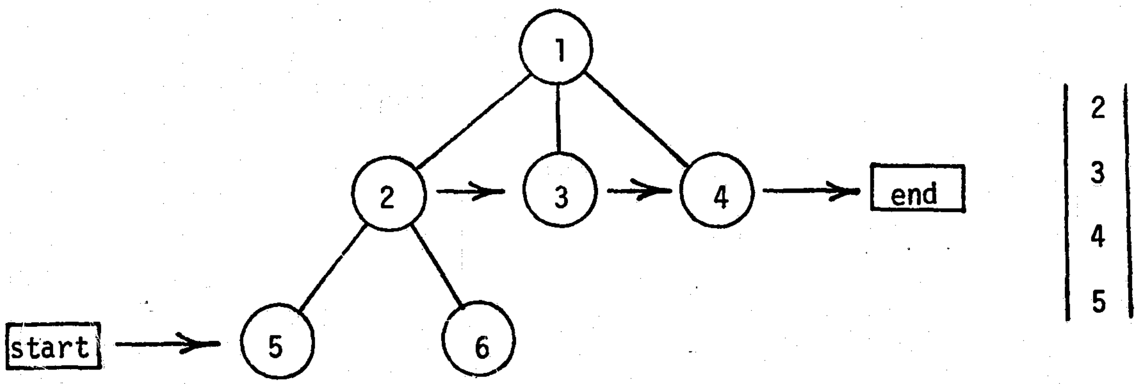


3. Since node 5 is a first son, time-link it to the predecessor of the node at the top of the stack (start).

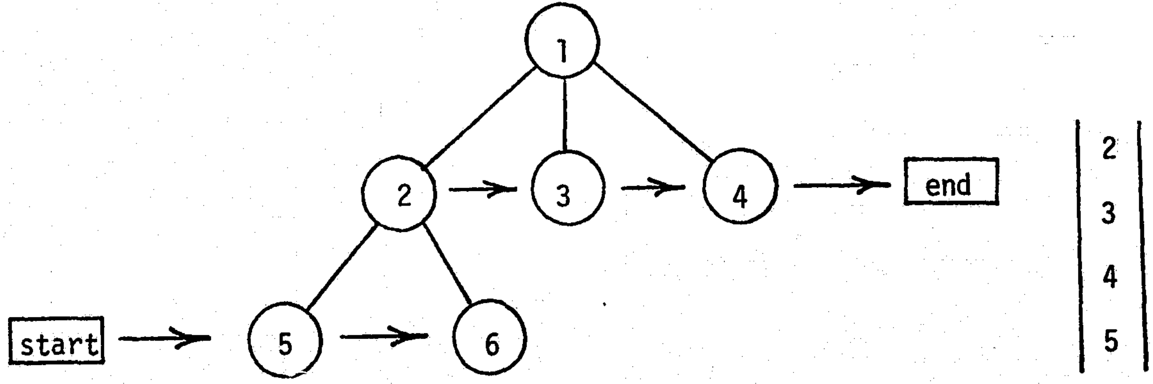


4. Place son on the bottom of the stack.

5. Create the next son of node 2 (node 6) noting that it is the last son.

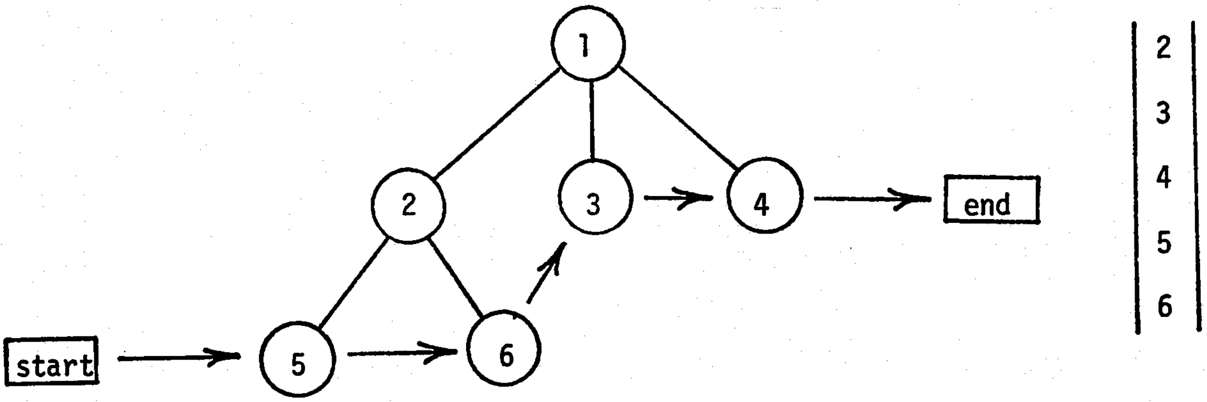


6. Time-link node 6 to the node at the bottom of the stack (node 5)



7. Place son (node 6) on bottom of stack.

8. Time-link son to successor of node at the top of the stack (node 3).

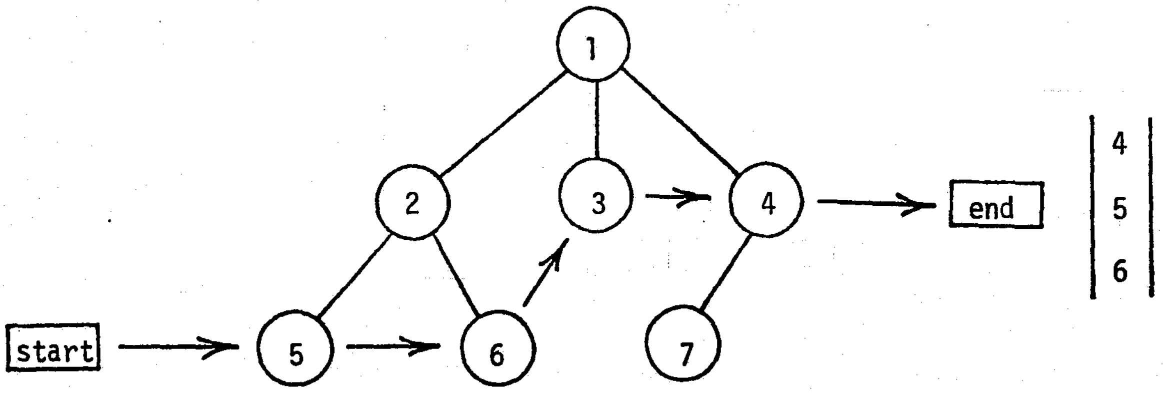


9. Exit to outer loop; pop top of stack; expand next node (node 3). Notice that at this point, a complete plan exists in the net for accomplishing the initial command. This property is always true at the end of each node expansion cycle.

10. Node 3 has no sons.

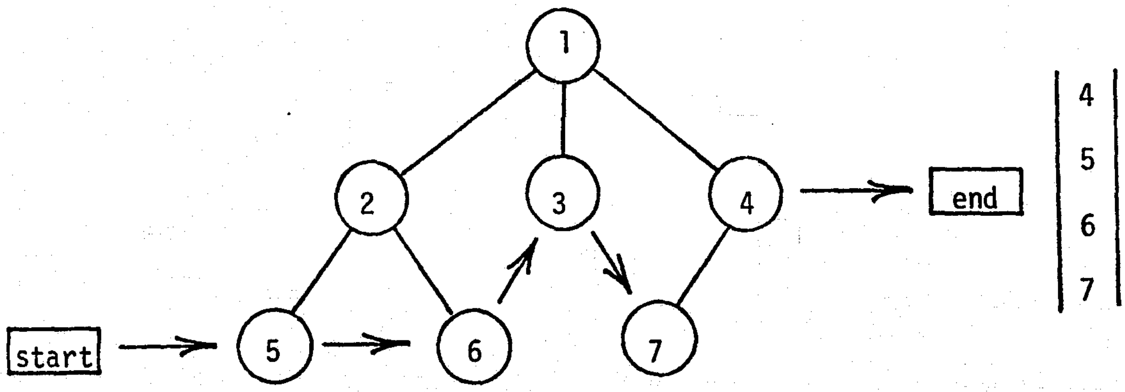
11. Exit to outer loop; pop top of stack; expand node 4.

12. Create first son (node 7) of node 4.

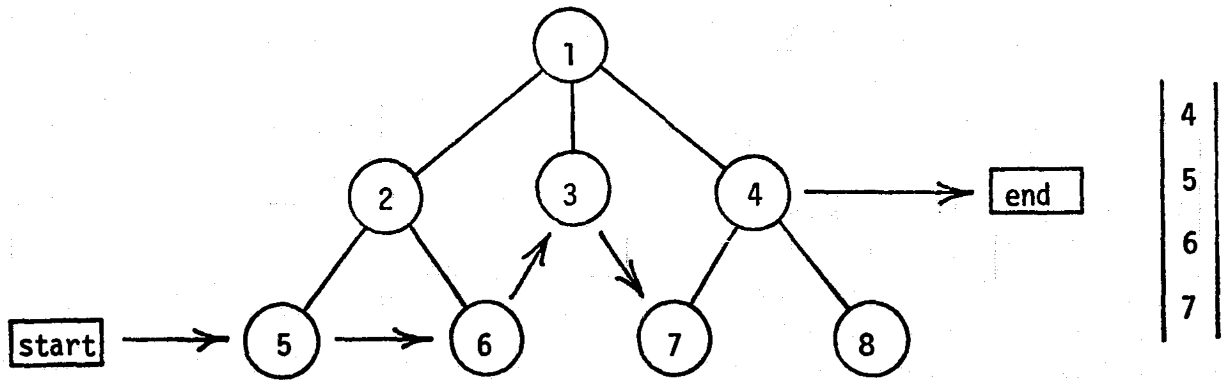


13. Time-link son to predecessor of node at top of stack (node 3).

14. Place son on bottom of stack.

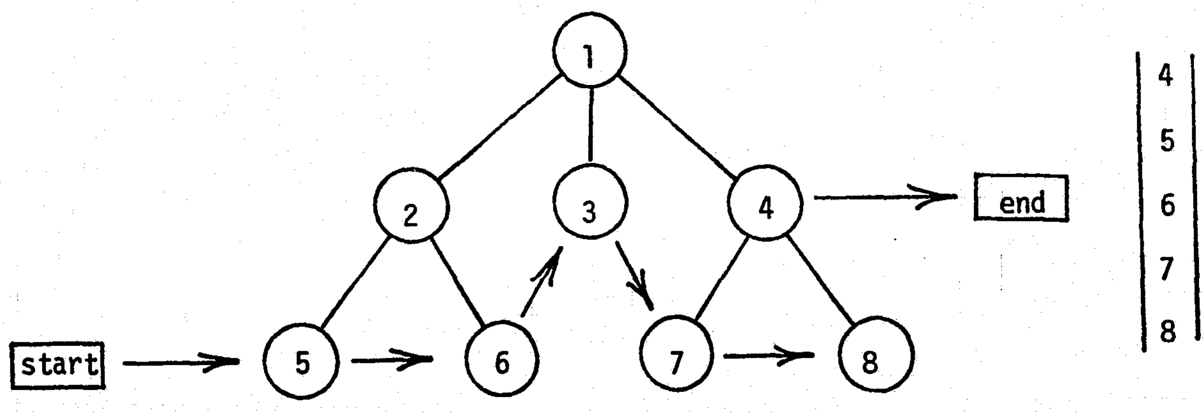


15. Create last son of node 4 (node 8).

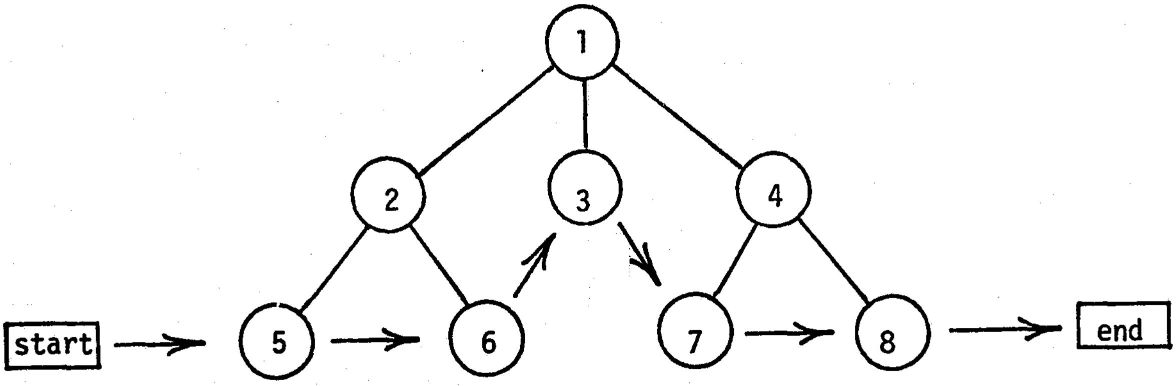


16. Time-link node 8 to node 7.

17. Place son on bottom of stack.



18. Time-link son (node 8) to successor of node at top of stack (end).



19. Since nodes 5 through 8 have no sons (by assumption), the stack will be popped until it is empty. This completes the procedural net expansion.