Expert Systems for the Scheduling of Image Processing Tasks on a Parallel Processing System

A Thesis Proposal

Submitted to the Faculty

of

Purdue University

by

Francis J. Weil

December 1986

This Research was supported by the Air Force Office of Scientific Research under grant F49620-86-K-0006 and the Rome Air Development Center under contract number F30602-83-K-0119.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 89-1192 F49620-86-K-0006
6a. NAME OF PERFORMING ORGANIZATION: PURDUE UNIVERSITY; 6b. OFFICE SYMBOL (If applicable):	7a. NAME OF MONITORING ORGANIZATION: AFOSR/NE
6c. ADDRESS (City, State, and ZIP Code): SCHOOL OF ELECTRICAL ENGINEERING, WEST LAFAYETTE, IN 47907	7b. ADDRESS (City, State, and ZIP Code): BUILDING 410, BOLLING AFB, DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR; 8b. OFFICE SYMBOL (If applicable): NE	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F49620-86-K-0006
8c. ADDRESS (City, State, and ZIP Code): BUILDING 410, BOLLING AFB, DC 20332	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. DARPA; TASK NO; WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification) ~~THE MAPPING OF PARALLEL ALGORITHMS TO RECONFIGURABLE PARALLEL ARCHITECTURES/CONTRIBUTED PAPERS 36/AN APPLICATION OF TENSOR THEORY TO 3-D SHAPE ANALYSIS/ THE PASM PARALLEL PROCESSING SYSTEM: DESIGN, SIMULATION, AND IMAGE PROCESSING APPLICATIONS~~/

12. PERSONAL AUTHOR(S) SEE BACK FOR 11.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
FINAL	FROM 01JAN86 TO 31DEC89		

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This Final Report DARPA Contract No. F49620-86-K-0006 is in the form of ~~published papers,~~ a masters thesis, a Ph.D. thesis, and a Ph.D. thesis proposal all supported in whole or part by the contract.

UNCLASSIFIED

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
GOLES	(202) 767-4931	NE

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

11. ~~A NOTE OF TWO-DEMENSIONAL LANDMARK-BASED OBJECT RECOGNITION/ON LANDMARK-BASED SHAPE ANALYSIS/THE PASM PARALLEL PROCESSING SYSTEM: HARDWARE DESIGN AND INTELLIGENT OPERATING SYSTEM CONCEPTS~~/EXPERT SYSTEMS FOR THE SCHEDULING OF IMAGE PROCESSING TASKS ON A PARALLEL PROCESSING SYSTEM

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The algorithms used in image processing are becoming longer and more complex as researchers strive to create vision systems whose performance rivals that of the human's. The size and complexity of these algorithms, however, generally do not allow them to be run in 'real time' on any sequential (Von Neumann) machine.

Image processing algorithms tend to be highly parallel in nature. One can hope, therefore, that the recent advances in parallel computing will bring significant speed-ups in the execution times of image processing algorithms. However, it is usually the case that image processing systems are extremely computationally intensive. Even with speed-ups brought about by parallel computers, there is a demand for an advisory system that optimizes the execution time of image processing algorithms.

A reasonable goal for such a system is as follows. Given a list of all the subtasks that need to be run for a given image processing task, produce an initial schedule and configuration and then adjust the schedule and configuration during runtime based on the current configuration and intermediate processing results.

The proposed work will proceed towards this goal on several fronts. One necessary component of such an advisory system would be a knowledge base

that incorporates the image processing algorithms and the information about these algorithms. Thought must also be given as to what is the minimum information needed to schedule an algorithm not contained in the knowledge base.

The advisory system and the parallel processor need to be in nearly constant communication. The processor's communication overhead, however, must be kept at a minimum to allow it to complete its processing tasks with minimal delay. It must therefore be determined what information the advisory system needs from the processor to make accurate execution-time decisions, how this information can be read, and how often.

The advisory system itself will be constructed in the form of an expert system. The expert system will have two parts. The first part will determine the initial schedule and configuration and pass these to the parallel processor's operating system. The second part will monitor the progress of the algorithms and dynamically update the schedule and configuration based on any new information received.

The proposed work will be based on the PASM parallel processor and the CLIPS expert system shell.

CHAPTER 1
INTRODUCTION

The field of image processing (and its subset, image understanding) is undergoing a period of rapid growth. The algorithms used by this field are becoming longer and more complex as researchers strive to create vision systems whose performance rivals that of the human's. The size and complexity of these algorithms, however, generally do not allow them to be run in 'real time' on any sequential (Von Neumann) machine.

Image processing algorithms tend to be highly parallel in nature due in part to the two dimensional structure of the images. One can hope, therefore, that the recent advances in parallel computing will bring significant speed-ups in the execution times of image processing algorithms. However, it is usually the case that image processing systems are extremely computationally intensive and that computer resources are at a premium. Even with speed-ups brought about by parallel computers, there is a demand for an advisory system that optimizes the execution time of image processing algorithms [Delp85].

The ultimate goal of such an advisory system built for a reconfigurable parallel processor can be stated as follows. Given a general image processing task, produce an initial subtask list, schedule, and configuration for the processor and continually update them based on intermediate results (noise measures, number of regions, etc.). This goal is somewhat ambitious considering the broad range of image processing tasks currently in use and the limited theoretical knowledge available.

A more reasonable statement of the goal would be as follows. Given a list of all the subtasks that need to be run for a given image processing task, produce an initial schedule and configuration based, in part, on the statistics and properties of the input image (noise, etc.). The advisory system would then adjust the schedule and configuration during runtime based on the current configuration and intermediate processing results.

The proposed work will proceed towards this goal on several fronts. One necessary component of such an advisory system would be a knowledge base that incorporates the image processing algorithms and the information about these algorithms. Many decisions have to be made about this knowledge base including which algorithms should be included and by what information these algorithms should be classified. Thought must also be given as to what is the minimum information needed to schedule an algorithm not in the knowledge base and, if no information is available, how it can be worked into the schedule and with what configuration.

The advisory system and the parallel processor need to be in nearly constant communication. The processor's communication overhead, however, must be kept at a minimum to allow it to complete its processing tasks with minimal delay. It would be counterproductive to have the control unit halt all active processors to get some trivial piece of information not really needed to make accurate decisions. It must therefore be determined what information the advisory system needs from the processor to make accurate execution-time decisions, how this information can be read, and how often.

The advisory system itself will be constructed in the form of an expert system. The knowledge needed to make valid decisions can then be encoded as facts and rules which can be easily modified. The expert system will have two parts. The first part will determine the initial schedule and configuration and pass these to the parallel processor's operating system. The second part will monitor the progress of the algorithms (by communicating with the operating system) and dynamically update the schedule and configuration based on any new information received.

The proposed work will be done using the PASM [Sieg81] parallel processor as the model for the reconfigurable parallel processor. The expert system will be written using the CLIPS (C Language Intelligent Production System) expert system language developed at NASA.

Section 2 provides a general overview of image processing tasks. Section 3 describes the PASM processor and its relation to the advisory system. Section 4 presents a discussion of expert systems and why they are of use for this class of problems. Section 5 discusses the knowledge base that will be used by the advisory system. Section 6 is a proposal to investigate the full advisory system (expert system and knowledge base) and describes the testbed system to be used in verifying the work.

CHAPTER 2
IMAGE PROCESSING TASKS

The field of image processing has existed for over twenty years. Credit for designing and implementing the first computer vision system is usually given to L. G. Roberts [Robe65]. With the advent of computer technology and the desire to make intelligent machines, the application of computers to vision seemed like a natural step. Two major facts soon became apparent. The first was that computers could easily perform vision tasks that humans found difficult or impossible (determining overall brightness, specific color value, etc.). The second fact was that those tasks that a human could do with little or no thought (or effort) where extremely complex on a computer (edge detection, stereo correlation, etc.).

Unfortunately, most vision systems require the tasks that a human would feel were trivial. Tasks such as navigation, target recognition, and depth perception through stereo vision are an integral part of most practical vision systems.

2.1. Terminology

At this point, a few distinctions between terms need to be made. There are two general terms used to denote imaging procedures: image processing (also known as picture processing) and image understanding. Image understanding is a part of image processing that falls under the category of high-level processing [fig. 2.1].

Low-level routines are those that do straightforward and relatively simple processing. They are generally of a "pixels in, pixels out" nature. Examples of low-level routines would be Laplacian filtering, the Prewitt operator [Prew70], Robert's cross [Robe65], the Hueckel operator [Huec71, Huec73], gray value thresholding, and Gaussian smoothing. Mid-level routines tend to deal more with structures in the image than with individual pixels. These routine are generally more complicated and time consuming than low-level routines and include such processes as region segmentation

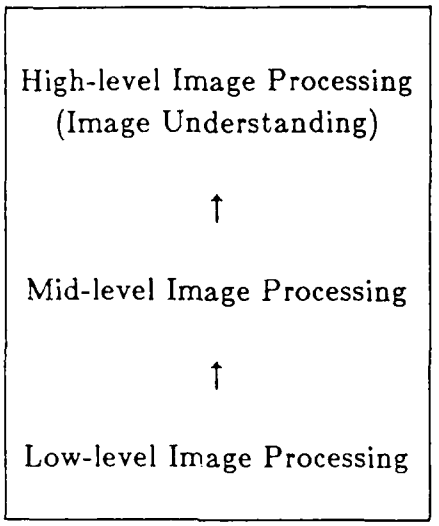

Figure 2.1 Digital Image Processing Hierarchy

and edge linking. High-level routines are those that try to build a consistent interpretation of the input image. The interpretation is usually either in the form of an internal representation (so that comparisons can be made or questions can be answered) or in the form of a natural language description. Image understanding programs are often written as expert systems since there is no general algorithm for the consistent labeling of segments in an image. However, it is possible to write high-level routines with no native intelligence. For example, a natural language description of a scene could be obtained using a statistical pattern recognition approach.

There are no rigid rules used to make the distinction in processing levels. A program of one level is usually assumed to include routines from the lower levels. That is, a high-level program uses as input an image that was processed by both mid- and low-level routines. These classifications, however, are largely cosmetic and serve only as a common basis for discussions.

Within any given level of image processing routines, there are tasks and subtasks. A task is the overall processing that is desired for the given application. An image understanding task might be to find the tanks in the infrared data. It does not specify exactly how the recognition is to be accomplished, only that that is what is desired. As previously mentioned,

mid- and low-level image processing tasks might be region segmentation and Laplacian filtering respectively. Obviously, there can be subtasks within subtasks and the labeling of a procedure as a task or subtask depends on the current processing context. The lowest level of subtasks is the actual code for the processing. All higher levels of subtasks exist only as part of the overall plan for the processing.

2.2. General Image Characteristics

A digitized image consists of a two-dimensional array of intensity values. The image, therefore, has been quantized in two ways. First, the spatial resolution (and the array size) is determined by the number of sample points per square inch that the input device (usually a camera) has and dictates the amount of detail that is retained by the digitized image. Typical resolution is 256 by 256 pixels while high resolution images can be several thousand pixels on a side. Second, the size of the word used to store each pixel value determines how many different brightness values (or gray levels) can be distinguished. A typical pixel value is stored as an 8-bit quantity giving 256 different gray levels. Quantizing each pixel value to 16 bits gives a resolution of 65,536 gray levels. As is apparent from the above data, one image can present a large amount of data for storage and processing. A 256 by 256 image quantized to 8 bits occupies 65,536 bytes (64 Kbytes) while a 2048 by 2048 color image quantized to 16 bits occupies 25,165,824 bytes (24 Mbytes).

Digitized images can be classified in one of three categories: binary, black and white, or color. Binary images are those that have intensity quantized to only one bit (either on or off). Black and white images are those that have intensity quantized to an arbitrary number of bits (with b bits corresponding to 2^b gray levels) but preserve no color information. Binary images could be considered a subset of black and white images but are considered separately because of the large differences in processing techniques used to analyze and manipulate them. Color images consist of three separate images (usually one each for red, green, and blue) which are combined to give the composite picture.

Binary images take the least amount of time to manipulate since many of the operations can be simplified as a result of the one bit quantization. Multiplications, additions, and comparisons are reduced to simple Boolean operations. However, this simplification is at the expense of the information contained in the intensity values such as some region boundaries. Color images are basically a set of three black and white images which are

processed independently. The three images are combined (added) on a color display device in the same way a television CRT operates. Color images convey the most information since two different colors with the same gray scale value are indistinguishable in a black and white image. The choice of the resolution and the color information is highly dependent on the current application (or task).

2.3. Image Processing and Parallelism

Many image processing tasks readily lend themselves to parallelism. In other words, the structure of the image itself combined with the type of processing desired suggest a natural parallel programming approach. Parallelism can most easily be achieved from two types of image processing algorithms: those that run the same procedure on successively smaller regions and those that repeatedly apply some small region operation (often a 3 by 3 window) to the whole image.

The most common procedure that operates on successively smaller regions is region splitting. In this process, general regions are separated from the image based on gray scale value, color, detected edges, etc. Each individual region is then processed the same way until either there are no more regions to process or the regions are too small to be split further. The most obvious approach to parallelism here is to assign one processor to each region. As more regions are formed by the process, they are assigned to new processors. In this way no processor has a backlog of regions to process and the analysis can proceed with minimum delay. In this case, the processors are said to be running in MIMD mode [Flyn66] (Multiple Instruction stream - Multiple Data stream) since each processor executes its own set of instructions on its own data.

Algorithms that operate over the range of the whole image are very common in low-level processing. Many operators such as Robert's cross and the Sobel operator operate on 3 by 3 windows. Other operations, such as Gaussian smoothing and edge masking, use larger windows such as 5 by 5 or 7 by 7. An odd-sized window is chosen so that there is a clearly defined center pixel. An operator such as thresholding processes each pixel independent of its neighbors (assuming that there is *a priori* knowledge of the threshold value). Each processor operates on its own portion of the image, possibly communicating with adjacent processors depending on how shared memory is set up. In this case the processor is said to be running in SIMD mode [Flyn66] (Single Instruction stream - Multiple Data stream) since all

processors execute the same instruction at the same time on their own set of data.

Even algorithms that do not suggest a natural form of parallelism can be structured so that parallel code can be generated. For these reasons and the fact that sequential image processing algorithms tend to be slow, there has been a great deal of interest in parallel processors and algorithms for image processing [Duff82] [Kryg76] [Rice85] [Rohr77] [Sieg81].

CHAPTER 3
THE PASM PARALLEL PROCESSOR

The target processor for the proposed expert system is the PASM (PArtitionable SIMD/MIMD) parallel processing system being developed at Purdue [Sieg81] [Kueh85] [Sieg86]. PASM will be able to be configured into a number of independent virtual machines of different sizes and modes (i.e. the system might be running with 4 partitions: two with 32 processors each in MIMD mode, one with 64 processors in MIMD mode, and one with 128 processors in SIMD mode). One of the design goals of PASM is to develop a system that can function as an environment for studying the use of parallelism in applications such as image processing. When completed, PASM will utilize 1024 separate processors in its parallel computation unit.

3.1. General Description

Figure 3.1 provides a diagram of the PASM system prototype. The hardware and software of PASM are each built up in levels so that the engineer or programmer need not be concerned with inappropriate details.

3.1.1. Hardware

The PASM system can be described on three different levels: the hardware level, the interrupt level, and the reconfiguration level [Schw86].

The hardware level model consists of the physical components and connections and does not directly concern the end user.

The interrupt level model contains the hardware information needed by the system programmer such as interrupt handling and memory management. This model does not contain all the physical level details of the hardware model, but still contains more information than needed by the end users.

On the reconfiguration (or highest) level (see figure 3.2), the PASM system consists of a system control unit (which coordinates the overall function of the system), a parallel computation unit (which contains the

individual processing elements and the communication network), a memory storage system (secondary storage for the parallel computation unit), a memory management system (which controls file transfers between the processors and the memory storage system), control storage (containing the programs for the micro controllers), and micro controllers (which control the activities of the processors).

Each processing element (or PE) of the parallel computation unit consists of a microprocessor, memory for storing data and code, and I/O controllers for communicating with the other PE's and the control units. All PE's can operate in both SIMD and MIMD modes. The mode of operation depends on what has been specified by the control unit for the current task.

The inter-PE communication in the parallel computation unit is accomplished through the use of an interconnection network. This network allows the transfer of data from one PE to any other PE. The network, along with routing tags on messages, also allows the system to form its independent partitions. The type of interconnection network that is being used in PASM is the Extra Stage Cube network [Adam82] which is a single-fault tolerant multistage cube.

Communication between the Purdue Engineering Computer Network (ECN) and PASM is done through the system control unit and the I/O processor in the memory management system. The user's terminal is connected to one of the host computers which provides the development and debugging environments. Tasks are sent from ECN to the system control unit for scheduling and execution. This scheme prevents overburdening the system control unit with user I/O. The memory management system I/O processor provides access to the storage and retrieval facilities of the ECN computers.

3.1.2. Software

PASM utilizes several different types of software. Each controller in the system (memory management, micro controllers, etc.) executes its own code. This code is transparent to the system user since it handles the details of data transfer, interconnection network configuring, and other tasks the user does not need to know about. Logically, some of the functions of the system control unit are to run the scheduler part of the operating system and to control the high-level user interface. However, there is no reason that the scheduler must be run as part of the system control unit. It could be run on

a separate processor or even in a partition of the PE's. The final location for its execution will be determined by available resources and the need to eliminate any computational bottlenecks. The programmer or user needs to know what operating system commands are available for job control. The language used to implement a particular parallel algorithm is the software that affects the programmer the most. The programmer has to deal with the intricacies of the algorithm such as which mode to use, how the data will be divided among the PE's, and what data needs to be shared and with which other PE's. However, he or she does not have to know which physical PE is executing the instructions since the system can be in a virtual machine mode.

The proposed expert system will be a part of the operating system in the system control unit. Since the expert system's function will be to advise the scheduler as to the best configuration and schedule to achieve improved performance, it will have to be provided with enough information to make valid decisions. This requirement places an added burden on the user, but the extra information has to be determined only once for a particular algorithm or task. The extra information needed is discussed below.

3.2. Dynamic vs. Startup Advice

Two types of information will be supplied to the parallel processor's operating system by the proposed expert system. The first is startup advice (which is supplied before the program begins its execution on the system) and the second is dynamic advice (which is periodically supplied as the program executes).

The startup advice consists of a best assessment for the initial configuration and schedule of the parallel processor based on information supplied by the user. The information needed from the user is a dependency graph (or precedence graph) which explicitly states the necessary time ordering of the subtasks and indicates all decision (or branching) points in the algorithm. Note that the term "graph" is used instead of "tree" since by definition a tree cannot have more than one path joining any two given nodes. This scheme assumes that the task (for the purpose of this research, the task will be in the image processing domain) can be outlined in terms of pre-written routines that perform basic functions. These routines will be classified by their execution characteristics. From the supplied information and the data already known about the pre-written routines, the expert system can decide which version of certain routines to use, if computations can be overlapped in some way, if large data transfers can be avoided, and

the best configuration for the PE's.

Once the program has begun its execution, the schedule and configuration can be updated to account for weak initial conclusions caused by decision points in the code and unknown execution times. These updates are the dynamic advice given to the system. As the program is running, the state of the execution must be monitored. As PE's become available in overlapped operations, new processes will need to be started up if possible. As branching points in the code are reached, the proper path in the code has to be determined and a schedule must be generated for that path.

Many algorithms use an unknown number of processors and take an unknown amount of time to complete. For example, region segmentation is often written as a recursive process operating on successively smaller areas of the image. It is not possible to determine *a priori* the time and processor requirements. The expert system will watch the execution of code such as region segmentation to avoid the situation where processors are idle when they could be starting the execution of the next routine.

3.3. Information Needed by the Expert System

The information needed by the expert system to make scheduling and configuration decisions falls into 3 classes. The first type of data needed is the dependency graph for the routines used in the algorithm. From this user-supplied graph, the expert system learns of the routines that may be executed by the algorithm as well as all possible paths that the execution can follow. Without this information, no speedups from overlapped execution or minimized data transfers could result since the scheduler would have no means of determining which routines depended on data from other routines. Unless the expert system knows which routines can be run independently of other routines, it would have no choice but to assume that dependency exists at each step. Dependency among routines generally precludes their being run simultaneously. One way for the user to specify this information is in the form of IF-THEN-ELSE constructs. All decision and branching points in the code, along with the routines that would be executed under each possible condition, can be specified by nesting to arbitrarily deep levels these constructs. An example is presented at the end of chapter 5.

The second type of data required by the expert system is the execution characteristics of the pre-written algorithms. If nothing were known about the routines to be executed, no advanced planning would be possible.

Chapter 5 discusses this problem in more detail.

The last type of data needed by the expert system is information concerning the state of the parallel processing system hardware and information about the routines currently being executed (i.e. runtime information). There are many factors that influence the scheduling and rescheduling of algorithms. Some of these factors are the current configuration, reconfiguration overhead, the number of PE's idle in each partition, the data contained in each PE (and what form it is in), and derived information (results from processing steps that determine branching paths). The number of PE's idle in a given partition can be quite large in some cases. For example, if the process currently running in the partition spawns new processes, then towards the end of its execution there will likely be idle PE's as the spawned processes terminate. Also, for routines that have execution times heavily dependent on data characteristics, an initially large number of PE's may be allocated to a partition. As the routine executes, it may become apparent that many PE's are idle due to some data property such as low noise power.

Knowledge of the current configuration is required since it takes time to reconfigure the system. If two versions of a routine to be executed are in the knowledge base and one can be run in the current configuration, it would most likely be advantageous to use that routine. The reconfiguration overhead (how long it would take to restart the calculations in a different configuration) limits the time savings gained by changing the system when other data indicates it is advantageous to do so. The number of PE's idle in each partition helps determine if reconfiguration will free enough PE's to start the execution of another routine. Knowledge of the contents of individual PE's can reduce the amount of overhead needed to store and load data during the execution of an algorithm. Suppose that there are two routines that are to be run one after the other. If the second routine requires the data that the first produces, both routines have the data allocated among local PE memories in the same way, and both use the same number of PE's, then it would be wasteful to store the data after the first routine finishes and then to read it right back in at the start of the execution of the next algorithm. Derived information must be obtained by the expert system so that a schedule for the proper path after a branching point can be determined. Unless all paths at a branching point have the same execution characteristics, a schedule for that part of the algorithm cannot be computed beforehand.

The expert system also needs to be aware of hardware dependent information such as which configurations are possible, the total number of PE's available in the system, and the overhead of obtaining information on a specific PE. The proposed expert system would be tailored to the PASM parallel processing system, but the general principles will be extensible to any partitionable parallel processor. In this way, as hardware changes the expert system does not have to be restructured. Figure 3.3 provides a schematic overview of the information needed by the scheduling expert system.

Since the expert system will be a part of (or a companion to) PASM's scheduler in the system control unit, the communication of new schedules and configuration does not present a problem. The expert system produces an internal representation of the desired schedule and configuration. Putting this information into a form usable by the operating system reduces to a simple formatting problem.

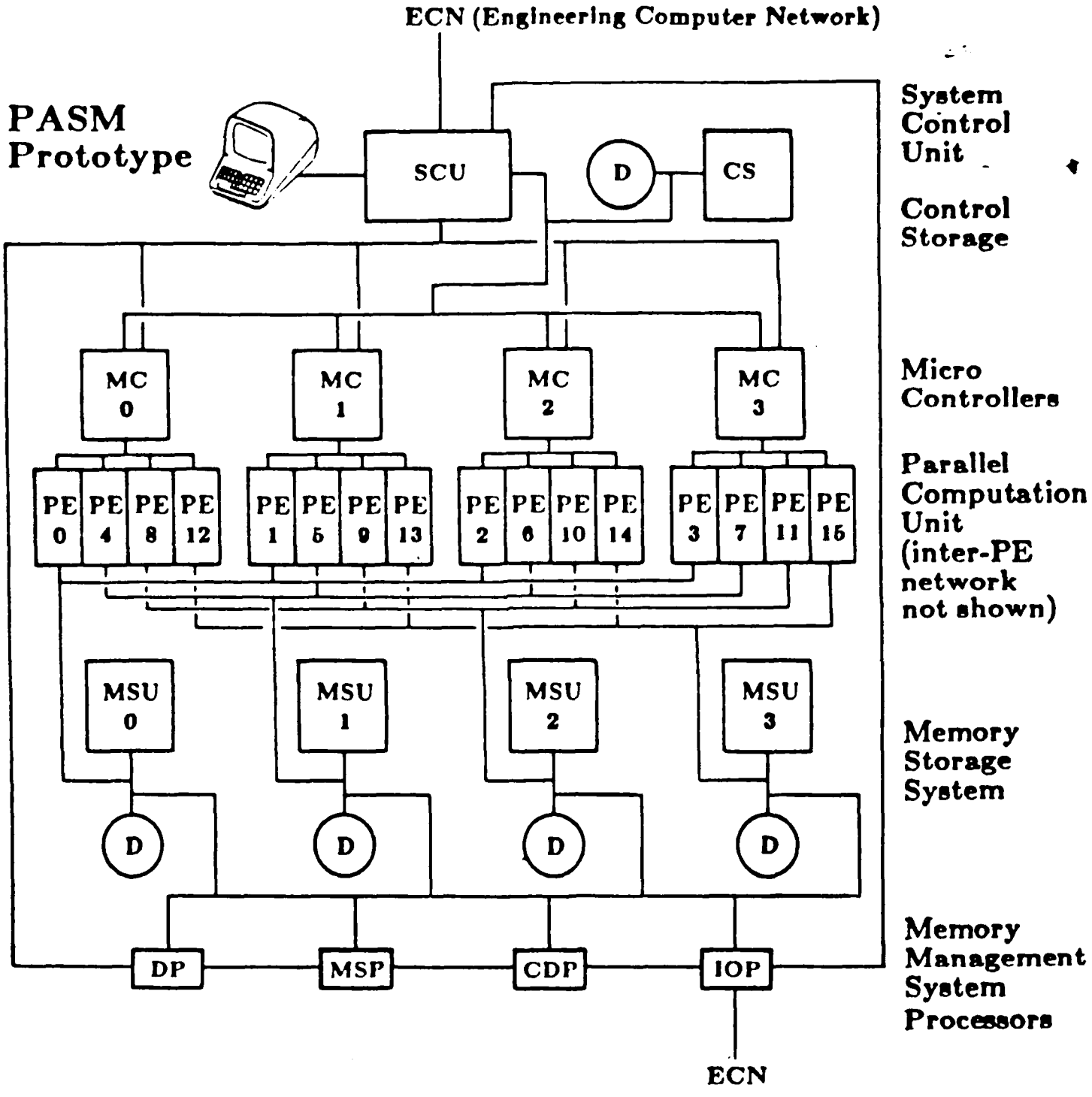

Figure 3.1 The PASM Parallel Processing System Prototype

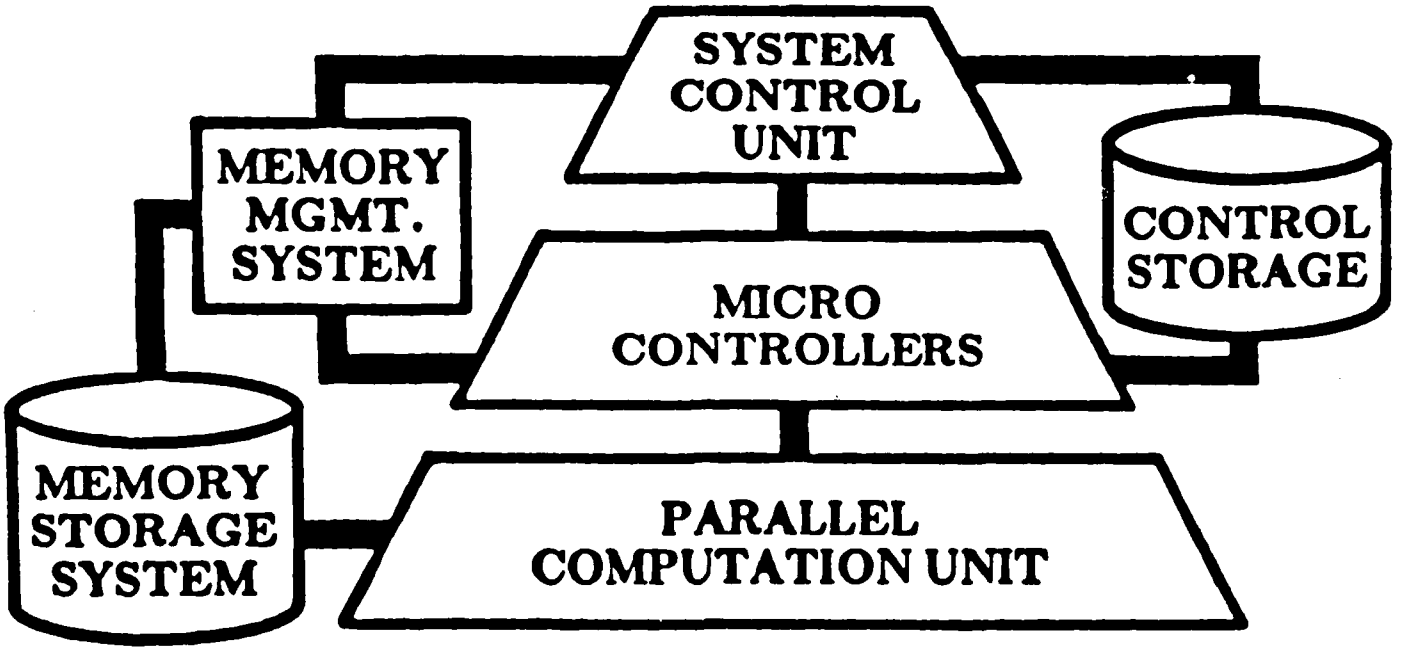

Figure 3.2 PASM Block Diagram

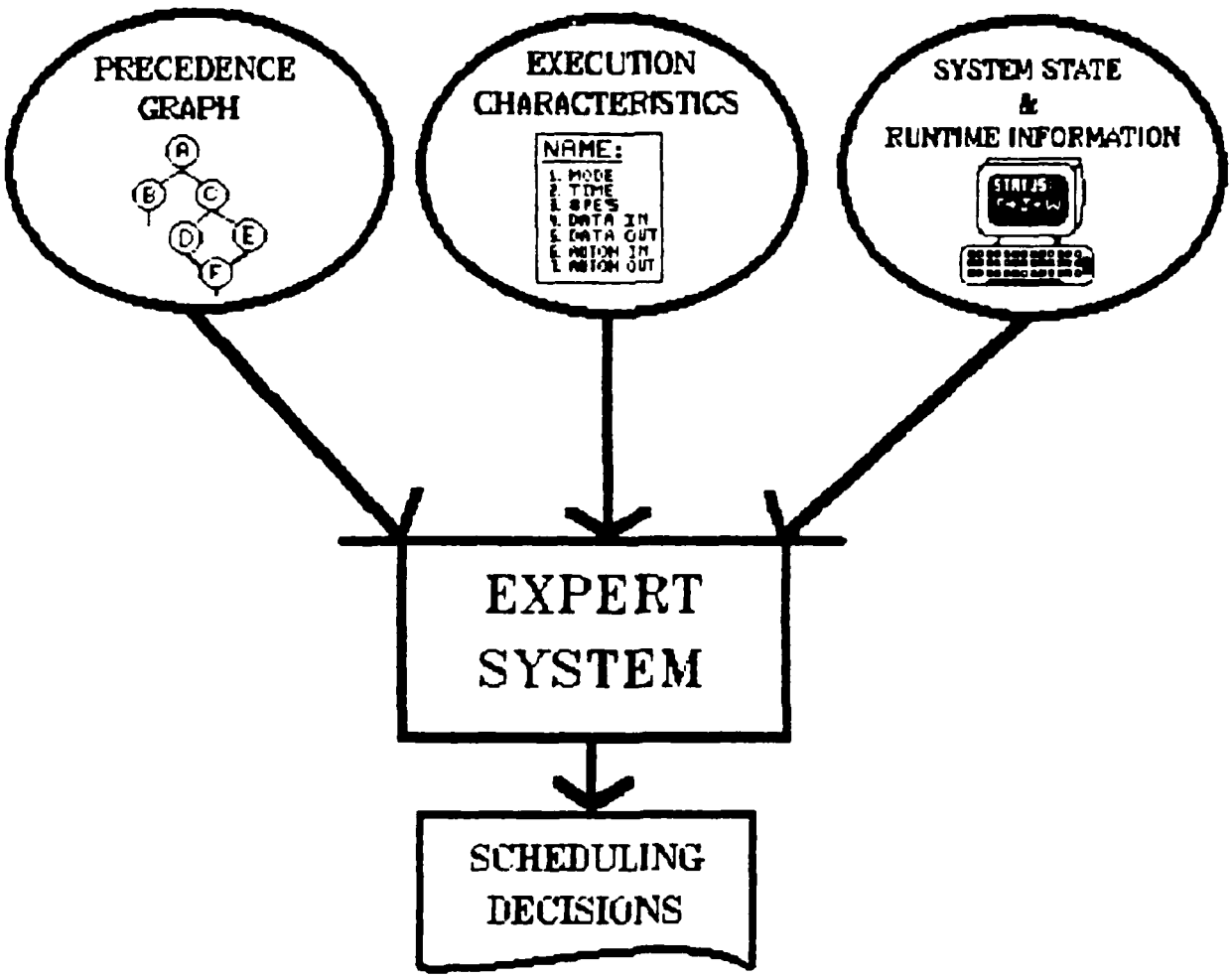

Figure 3.3 Expert System Information Diagram

CHAPTER 4
EXPERT SYSTEMS

Expert systems are an attempt to codify human expertise in some specific domain in the form of a computer program. Successful expert systems have been built in the areas of medical diagnosis (MYCIN [Shor76]), speech understanding (Hearsay-II [Erma80]), organic chemistry (Dendral [Lind80]), mathematics (Macsyma [Mart71]), and many other fields.

4.1. General Description

What distinguishes an expert system (ES) from an ordinary program? This question is still not decisively settled, but a few guidelines for and characteristics of ES's can be stated to help make the distinction.

An essential aspect of expert systems (as the name implies) is the embodiment of some expertise. The expert's knowledge is coded in the form of facts and rules. A somewhat restricted definition of 'expertise' is needed if it is to be of any help in categorizing ES's. Obviously a program that can only add the numbers 3 and 5 to get a result of 8 could be termed an expert in adding 3 and 5, but this classification is of minimal help. However, a program that knows the rules of addition (i.e. when given the fact that some number was added to 3 to get 8 can deduce that the unknown number was 5) is a good candidate for being considered an expert system. The difference in the two examples above is that the first deals with a specific example from a given domain and the second deals with a rule about the domain itself. In other words, an expert must embody knowledge about a domain and be able to deduce facts about objects within the domain that are not explicitly dealt with in the knowledge base (that is, every possible case is not specified beforehand).

4.1.1. Facts and Rules

The knowledge in an expert system is embodied in facts and rules. Facts are statements or properties that are known to be true about the given

domain at the present time. Examples of facts in the field of astronomy might be:

The sun is more massive than the earth.
The earth is more massive than the moon.
The earth is a satellite of the sun.
The moon is a satellite of the earth.

Rules are statements of relationships between facts and methods of manipulating facts. Rules are used by the ES to derive new facts and to verify hypothesized facts. Example of rules in astronomy might be:

1)

IF A is more massive than B
AND
B is more massive than C
THEN
A is more massive than C.

2)

C is a satellite of A IF
C is a satellite of B
AND
B is a satellite of A.

From these facts and the first rule the ES can deduce that the sun is more massive than the moon. From these facts and the second rule the ES can deduce that the moon is a satellite of the sun. No explicit mention of these facts was ever made in either the facts or the rules. If the fact "The earth is heavier than Mercury" is added, the new fact "The sun is heavier that Mercury" can then be deduced. The above example could be considered an expert system (albeit a trivial one) because it can use its rules and facts to derive totally new facts. By a similar method, an ES would be able to answer the question "Is the sun heavier than Mercury?" even though it does not know the answer explicitly when asked.

4.1.2. Inference Engines

All the facts and rules that could be collected would be useless unless there were some systematic method of manipulating them. This method is called the inference engine. The inference engine is in charge of determining what the current goal is, what rules have to be applied to achieve the goal, and when the goal is satisfied. From the astronomy example above, if the ES were asked the question "Is the sun more massive than the moon?", the inference engine would set answering this question as its current goal. The fact list would be examined and no match would be found. The inference engine would then decide that if it could prove that the sun were more massive than something and that that something were more massive than the moon, the goal would be satisfied. By re-examining the fact list in light of this new goal, the facts that the sun is more massive than the earth and that the earth is more massive than the moon fit the pattern. This new goal is satisfied which in turn satisfies the original goal. This process is known as conflict resolution.

The inference engine is independent of the facts and rules. Different expert systems in totally different domains can be written to use the same inference engine, but the facts and rules must be written to conform to the syntax expected by the engine. The inference engine, along with tools that make the programmer's job easier (execution tracing, etc.), are often referred to as an expert system shell.

4.1.3. Forward vs. Backward Chaining

Inference engines are either forward or backward chaining (or sometimes a combination of the two) depending on whether the rules are data driven or goal driven respectively.

A forward chaining (or data driven) system uses the available facts and the rules to try to derive new facts until the goal is met. Forward chaining rules follow the form:

IF (Boolean condition set) THEN (new facts).

An example of this type of rule is the first one of the astronomy example above.

A backward chaining (or goal driven) system determines what facts have to be verified to validate the current goal and, if they are not present, looks for a rule that restates the goal as a set of other goals. Backward chaining rules follow the form:

(goal) IF (Boolean goal set)

The second rule of the astronomy example is an example of a backward chaining rule.

The choice of whether to use forward or backward chaining depends on the application. As a rule of thumb, if the ES has a large amount of facts and rules that can be combined in several ways, backward chaining should be used to avoid combinatorial explosion. For example, if one task of an ES is to classify a vehicle as being a truck or a car and there are hundreds of facts stored about the vehicle, it would be more efficient to see if the vehicle is a car (and if not, then if it is a truck) than to try to arrive at that conclusion based on all the facts. A rule like "IF it has four wheels AND it has an engine THEN it is a motor vehicle" would be of little use in the classification.

Forward chaining should be used when the facts can only be combined in a limited number of ways and it would be inefficient to rederive facts for every possible goal. For example, if there are 10 facts about a widget in the fact list and the ES is trying to place the widget in one of 1000 classes, it would be wasteful to try each category to see if the widget belongs there. A better method would be to look at the facts and use the rules to restrict the classes to which the widget might belong (e.g. a rule like "IF the widget is red or green THEN it is type 17, 34, 397, 668, or 999").

Most ES's can be run as either data or goal driven with a simple syntax change for the rules. How the inference engine applies (or 'fires') the rules rarely effects the logic of the rules themselves.

4.2. Expert Systems and Image Processing

There are many advantages to using expert systems for image processing problems. The non-determinism of image processing procedures is well suited to the structure of ES's. Since no program can determine beforehand all the properties of an image and therefore cannot predict which subtasks will be executed or how long they will take to complete, a flexible control scheme that can deal with uncertainties is needed. Expert systems, due to their rule-based structure, can easily set up and test different combinations of the facts as new data is acquired.

Another advantage of ES's in image processing stems from the facts that there are no set rules on how images should be processed and that no general algorithms exist to guide in the selection of image processing procedures. For example, if the current task is to do edge detection, there are many possible

ways to proceed. Possibilities include thresholding to a binary image and then running the Robert's Cross operator, running only Robert's Cross, using a Laplacian operator with edge thinning or edge linking depending on the results, using an edge mask, etc. The choice of which method to use depends on many factors, some of which cannot be determined *a priori*. Factors such as the amount of noise in the image, how critical it is to have good edges, what form the image is in (binary, color, etc.), and what algorithms had previously been run on the image contribute to determining which method to use. The ability of an ES to make choices based not only on the facts it started with but also on facts it can derive while running make it ideal for this situation.

Most image processing systems have many branching points in their algorithms. Different algorithms are run based on, among other things, the number of segments in the image, the amount of texture in a segment, and edge thickness. Each algorithm has its own execution speed and memory requirements, so *a priori* planning for minimum execution time is not possible. An expert system can revise its processing schedule to account for new data as it arrives. In this manner, a reduced (although usually not minimum) execution time can be achieved.

Since most image processing systems are built up piece by piece, an ES provides a good environment for the control scheme. As new algorithms are added or processing methods are changed, the programmer need only change (or add or delete) the rules and facts of the ES. A program in a conventional language such as C or Fortran usually has to be restructured every time a significant change is made if efficient coding is to be maintained.

Unfortunately, no method is without its disadvantages. Expert systems generally execute more slowly than conventional languages due to the resolution scheme and their interpreted nature. With the inference engine processing the facts and rules, the overhead is greater than with a purely compiled language.

Large memory requirements are common with ES's since rule information and new facts must be stored in a way that allows fast access (i.e. file access would be too slow). The other drawback of ES's is that even though they are good at manipulating symbolic information, number crunching requires extra effort. The typical way around this drawback is to provide an interface mechanism to a high level language such as C or Fortran so that computations can be done much faster than if they were

interpreted.

4.3. The CLIPS Expert System Shell

There are many expert system languages and shells available to the programmer, each having its own advantages and disadvantages. This research will use the CLIPS (C Language Intelligent Production System) shell [Lope86] developed at the Artificial Intelligence Section of NASA's Mission Planning and Analysis Division.

CLIPS has a C language interface so that complex math functions, file I/O, and operating system interfacing can be done with minimum effort. The CLIPS inference engine runs on BSD UNIX* which is used here at the Purdue University Engineering Computer Network. The source code for the shell is available so that additional features can be added as needed. CLIPS is forward chaining which, for most image processing tasks, should be more efficient than backward chaining.

4.3.1. CLIPS Description

CLIPS is a rule-based language that was designed to be highly portable and to be fairly fast at floating point calculations. The source code is written in the C language to allow easy portability to any machine with a C compiler. All internal structures used by the shell are allocated and disposed of dynamically so that the application is limited only by the amount of available memory.

CLIPS has four execution time options to aid in debugging:

Step Mode	allows the user to watch the rules fire one at a time by pausing the execution before the next rule is to fire.
Watch Mode	displays activation messages. Any time a fact is asserted or a rule is activated, a "==>" is printed following the name of the fact or rule. Any time a fact is retracted or a rule is deactivated, a "<==" is printed following the name of the fact or rule.

* UNIX is a Trademark of Bell Laboratories.

Display Facts & Agenda — will print the fact list and the agenda on the screen prior to the firing of each rule.

Display Function Table — will print the external C functions prior to the program execution.

4.3.2. Facts in CLIPS

Facts in CLIPS can be asserted or retracted during execution. An initial fact list can be given to the shell prior to the firing of any rules. CLIPS always starts with the fact "initial-fact" asserted. The syntax of an initial fact list is:

```
(deffacts NAME
     (FACT)
       :
       :
     (FACT))
```

where NAME can be any ASCII string and FACT is any set of words, strings, and numbers. An example of a fact list is:

```
(deffacts example
     (the sun rises in the east)
     ("water is wet")
     (cubed 3 27)
     (value pi 3.14159265))
```

The quotes around the second fact signify that the text for the fact is to be taken as a whole. This fact has only one component, namely "water is wet", and is totally different from the fact (water is wet). The advantage of using quoted text is that storage is more efficient and comparisons can be done more quickly. The disadvantage is a loss of flexibility in pattern matching for that fact. See the next section for a general discussion of pattern matching.

4.3.3. Rules in CLIPS

Once the initial facts are loaded, CLIPS begins processing the rules. The syntax of a rule is:

```
(defrule NAME
     #(salience NN)
     ?VAR<-(PATTERN)
        :
        :
     ?VAR<-(PATTERN)
=>
     (ACTION)
       :
       :
     (ACTION))
```

where NAME and VAR are ASCII strings and NN is a number. PATTERN and ACTION will be described below. The "#salience" term gives the rule a priority value. High priority rules are fired before those with lower priority whenever possible. If the "#salience" clause is omitted, the priority is assumed to be zero. All PATTERN clauses in the rule must evaluate to TRUE for the rule to fire and for the ACTIONs to be performed. The "?VAR<-" is prepended to a PATTERN if the fact that matches the PATTERN is to be retracted in one of the ACTION clauses on the right hand side (RHS) of the rule; it is not required otherwise.

All PATTERN clauses have an implicit AND condition surrounding them. A PATTERN can take on several forms, the simplest of which is a fact that must be matched. If the fact exists in the fact list, a TRUE is returned, otherwise the rule is skipped. A PATTERN can use a variable for the match with the fact list. A "?" will match any element in the fact and a "$?" will match zero or more elements in the fact. A variable name can follow the "?" or "$?" which will cause the variable to be instantiated with the match value within the scope of the rule. For example, suppose the fact "(the widget is purple)" existed in the fact list. The following PATTERNs would all match and return TRUE:

```
(the widget is purple)
(the widget is ?)
(the ? is ?)
(the ?object is ?color)
(? ? ? ?)
($?)
```

($? widget $?)
(the $?rest)

The following patterns would not match and would return FALSE:

(the widget is green)
(the widget is ? ?)
(the ?aa is ?aa)
(? ? ?)
(? the $rest)

A PATTERN can also contain the logical operators "&" (and), "|" (or), and "~" (not). For example, "(goodbye ~ruby&~tuesday)" will match any fact that has "goodbye" as its first element and anything except "ruby" or "tuesday" as its second element. More complex Boolean and arithmetic operations can be performed using the "test" predicate and the operators in table 4.1. The PATTERN:

(test (|| (> ?a 6) (&& (= 3 (/ ?a ?b)) (!= ?e ?f))))

will return true if ?a is greater than 6 or if ?a divided by ?b equals 3 and ?e does not equal ?f.

C functions can be called using the "&:" operator. Any non-zero return value indicates a value of TRUE. The PATTERN "(value ?x&:oddp())" will match a fact if the first element of the fact is "value" and the second element is an odd number. The value of the second element would then be bound to the variable "x" as a side effect of the test. Table 4.2 lists the builtin C functions that can be used in a PATTERN.

The ACTION clauses (on the RHS of the rule) can be one of four types: assert, retract, printout, or a C function. The "assert" clause puts a fact in the fact list. The syntax of "assert" is: (assert (FACT)) where FACT is as previously defined. The "retract" clause removes a fact from the fact list. The syntax of "retract" is (retract (?VAR ... ?VAR)) where "?VAR" is from the LHS of the rule as previously mentioned. The "printout" clause simply prints the specified data on the current display device. The syntax of "printout" is: (printout (ANYTHING)) where ANYTHING can be any set of variables or literals. A C function call can be made from the RHS by stating the name of the function and its argument list similar to the way it would be done from a C program except that spaces instead of commas separate the arguments. Facts can also be asserted from within a C function using the "assert" function.

Appendix A provides an example of a simple expert system implemented in the CLIPS language. The output of a full diagnostic run of the expert system is shown to further illustrate the concepts of pattern matching, rule firing, and the inference engine.

Table 4.1 CLIPS Test Operators

Operator	Operation
!	not
&&	and
||	or
=	equal
!=	not equal
>=	greater than or equal
>	greater than
<=	less than or equal
<	less than
/	division
*	multiplication
**	exponentiation
+	addition
-	subtraction

Table 4.2 CLIPS Predicate Functions

Function	Description
oddp()	checks if element is an odd number
evenp()	checks if an element is an even number
stringp()	checks if an element is a string
numberp()	checks if an element is a number
greaterp(arg1 arg2)	checks if arg1 > arg2
lessp(arg1 arg2)	checks if arg1 < arg2
gteqp(arg1 arg2)	checks if arg1 >= arg2
lteqp(arg1 arg2)	checks if arg1 <= arg2
equalp(arg1 arg2)	checks if arg1 = arg2

CHAPTER 5
IMAGE PROCESSING KNOWLEDGE BASE

The knowledge base is an integral part of any expert system. The rules and expertise embodied by the ES would be useless if there were no knowledge (in the form of facts) for the rules to be applied to. The proposed expert system's knowledge base will be comprised of image processing routines and knowledge about these routines.

5.1. Routines to be Included in the Knowledge Base

The routines to be included in the knowledge base (KB) are image processing routines of the low to mid level. These routines generally function as preprocessing steps in high-level algorithms. Since the ES will be working with tasks built with these routines as primitives, there is no need to include high-level routines in the KB. Any high-level algorithm (and some mid-level algorithms) would have to be tailored for the current task and would therefore be too application specific to be included in the KB.

The routines in the KB will be of a general nature rather than being written with a specific application in mind. Windowing operations (routines that replace a pixel with some function of the surrounding pixels) will have user selectable window sizes and pixel replacement functions. For example, a Gaussian filter routine might generate a 5 by 5 window with a standard deviation of 8 for one pass and a 49 by 49 window with a standard deviation of 17 for the next pass. Thresholding routines will have a selectable threshold value to accommodate images with various input characteristics, histogram routines will use selectable bin sizes, etc.

The routines themselves will cover the basic image processing functions. Examples of routines that will be included in the KB are edge linking (to link up disconnected edge pieces), edge thinning (to thin edges to single-pixel width), region segmentation (to split the image into homogeneous regions), thresholding (to transform an image into a binary image given some value for the 0/1 decision), histogram generation (to produce a histogram of the gray

levels of an image), and various windowing procedures (median filtering, Laplacian filtering, edge masks, Gaussian filtering, Robert's cross, etc.).

The KB will contain various versions of the above algorithms all lumped under the same general name. Each version of an algorithm will be classified by its run-time characteristics such as the mode used (SIMD or MIMD), the number of processing elements needed, the form of the data needed by the routine, and the form of the data produced by the routine. The user does not need to have any knowledge of the specific versions of the routines available. He or she only specifies which general routine is desired (e.g. a 5 by 5 Laplacian window) and the ES will choose the best version of the routine based on all the data available (other routines being used, etc.). This scheme allows the user-supplied image processing algorithms to be independent of the specifics of the KB. Once the user has written his or her application in terms of these general routines, he or she need not be concerned with new versions of KB routines. The only change that would be noticed is a decrease in the execution time of the algorithm. The user specified data is the same no matter how many versions of specific routines are in the KB. It follows, then, that as more versions of specific routines are added to the KB, the user becomes better off instead of becoming more burdened with detail.

5.2. Knowledge Base Algorithm Classification Scheme

Although the user does not need to be aware of all the complexities of the knowledge base, the expert system will largely base its decisions on information contained there (the ES will also use information about the target parallel processor). Ideally, the knowledge contained in the KB about the image processing routines would be an orthogonal set of execution-time characteristics. For a more thorough discussion of the topic, see [Jami85]. In a more practical view, there will always be some redundancy of information about the routines. For example, the number of processing elements that a routine uses (in either SIMD or MIMD mode) usually has a direct correlation with the expected execution time. However, one cannot cannot be inferred from the other. Also, the format of the data (i.e. 1 bit binary, floating point, etc.) affects the execution time of the routine. Again, equal execution times of two routines does not imply that the routines use the same data type and two routines that use the same data type do not necessarily have the same execution time. What is needed, therefore, is some set of classification data that is not overly redundant but that still encompasses all the data required

for scheduling decisions.

The KB algorithms will be classified on seven parameters. The first parameter is the mode of the algorithm (either SIMD or MIMD). The mode is important since it determines whether or not the parallel processor has to be reconfigured from SIMD mode to MIMD mode (or vide versa) for the given routine. This factor may or may not be significant depending on the amount of system overhead associated with the change. Reconfiguring the system takes time that cannot be used for processing and therefore should be minimized. The second parameter is the number of PE's that are required (or desired) for the routine. In general, the more PE's used by the routine, the faster the execution time (up to some maximum value for the routine). However, PE's used by one routine are obviously not available for another to use. Some compromise must be made between concurrent execution of routines and execution speed for individual routines. The third parameter is the expected execution time. This parameter is a function of the number of PE's used, the size of the data being operated on, the format of the data, the mode that the routine is operating in, and other factors. This information is important since it may be more efficient to run a slower routine without reconfiguring the system than it is to reconfigure the system and then run a faster routine. The fourth parameter is the format of the input data used by the routine. Binary operation are generally faster than floating point operation and should be used whenever possible. The fifth parameter is the format of the data generated by the routine. In reference to these two parameters, if data is in the form needed by the next routine to be executed, it is advantageous not to store the data when the first routine finishes and then reload it when the next routine starts. The last two parameters are input and output data autonomy. These parameters are indications of whether or not the routine uses data produced by other routines and whether or not the routine produces data used by other routines. Data autonomy is an indication of the amount of overlapped execution that can be achieved. These seven parameters represent the basic data needed by the expert system. Miscellaneous other information concerning specifics of certain routines will probably also be needed. Table 5.2 summarizes these classification parameters.

Several factors influence the execution-time characteristics of the parallel routines [Jami85]. The type of parallelism (dividing the data among PE's, dividing the algorithm among the PE's, or macropipelining), the module granularity (the amount of processing a PE can do independent of other

PE's), and the uniformity of the operations each PE must perform influence the choice of execution mode for the routine. The data granularity (the amount of data being processed by each PE), the static or dynamic nature of the routine (whether or not the routine spawns new processes), the inter-PE data dependencies (how much data one PE requires from another PE), the atomic data types (integer, floating point, etc.), and the module granularity affect the inter-PE communication requirements. The overall degree of parallelism that can be achieved and the atomic data types affect the execution time of the routine. The static or dynamic nature of the routine affects the scheduling of the task. However, if the number of subprocesses that will be generated by a routine is known beforehand, the subprocesses can be included in the overall scheduling task.

The user does not need to be directly aware of these parameters and factors, but a surface-level awareness of them would allow more intelligent choices to be made if there is more than one set of routines that can accomplish the current task. For example, suppose that there are two methods of processing a given task: (1) routine A followed by routine B followed by routine C or (2) routine D followed by routine C. On the surface, it would seem that choice (2) is better since there is one less routine that has to be run. However, it may be that case that choice (1) actually has a shorter execution time due to data autonomy and inter-PE data dependencies. Therefore, even though such detailed knowledge is not necessary, it can be an aid in making algorithm choice decisions.

5.3. Routines Not Contained in the Knowledge Base

It is not unusual for a specific task to have to do some "non-standard" processing. The user might know of some properties of the input image that allow some shortcuts in processing. The particular application might require some complex mathematical function to be applied to the image (summing the square root of the natural log of all pixel values, for example). Obviously, not all possible image processing routines can be included in the KB.

There is no obvious solution to the scheduling of non-KB routines, but there are two possible ways to handle this situation: blind scheduling and guided scheduling. Blind scheduling occurs when the user supplies no information about the routine to be executed. In the absence of any knowledge of the execution characteristics, the ES has no choice but to finish all routines in the precedence graph that have a higher temporal precedence

(that is, all routines that have to be run before the unclassified routine), execute the unclassified routine, and then start the execution from that point forward. This scenario is the worst possible in terms of execution time. The scheduler cannot take advantage of process overlapping and large amounts of the parallel processor's resources might be idle. Even though the routine might only use 32 processors and be totally data autonomous, no other routines could be executed concurrently.

A better, and in general more realistic, scenario is one in which the user can supply some rudimentary information about the unclassified routine. This scheme is termed "guided scheduling" since the ES has enough information to guide its scheduling decisions but it does not know all the desired information about the routine. Ideally, the same set of parameters used to classify a routine in the knowledge base would be supplied by the user. However, a minimal set that the user should supply is the execution mode (SIMD or MIMD), the number of PE's required, and whether or not the routine can operate autonomously on its input and output data. With this information, or as much of it that can be provided, the scheduling expert system can work the routine into the schedule to achieve some optimality. It is much better to know that the unclassified routine uses a maximum of 64 PE's and then allocate them all when the routine starts than to tie up all the resources because the number of needed PE's is unknown.

As a general guideline, it would be better to lump as many unclassified algorithms together in the precedence graph as is practicable. This scheme allows the expert system to schedule as large a block of known routines at one time as is possible. As more routines are scheduled as a unit, better choices as to routine selection can be made since more information is known about the overall processing environment.

5.4. Image Processing Example

To further illustrate the ideas presented above, an example of a small image processing algorithm is now given. For the purposes of this example, let the input picture be a 512 by 512 gray scale image with pixel values quantized to 8 bits and the algorithm be as follows:

Task: Split the input image into regions

1) Get edge information using the Laplacian operator.

2) Do Gaussian smoothing using a 10 by 10 window.

3) If the edges are thick

   3a) Then do edge thinning.

   3b) Else do edge linking.

4) Do recursive region segmentation.

The precedence graph for the above algorithm would be as in figure 5.1.

Once this precedence graph has been determined, it is a simple translation to IF-THEN-ELSE form. The major difference is that the routines have to be specified more precisely than "Get edge information using the Laplacian operator." The translated graph for this example would be:

```
IMAGE_SIZE(512,512,8)
LAPLACIAN
GAUSSIAN(10)
if EDGES(THICK)
    then {THIN_EDGES}
    else {LINK_EDGES}
RECURSIVE_REGION_SPLIT
```

The above list is the information the ES needs from the user to make its scheduling decisions. Note that even though there might be more than one version of an algorithm like RECURSIVE_REGION_SPLIT, the user need not be concerned with which one will be chosen. The ES will choose the version that best fits the schedule and configuration to optimize execution time. The IMAGE_SIZE statement shown above gives the user a shorthand notation for informing each KB routine of the physical characteristics of the stored image.

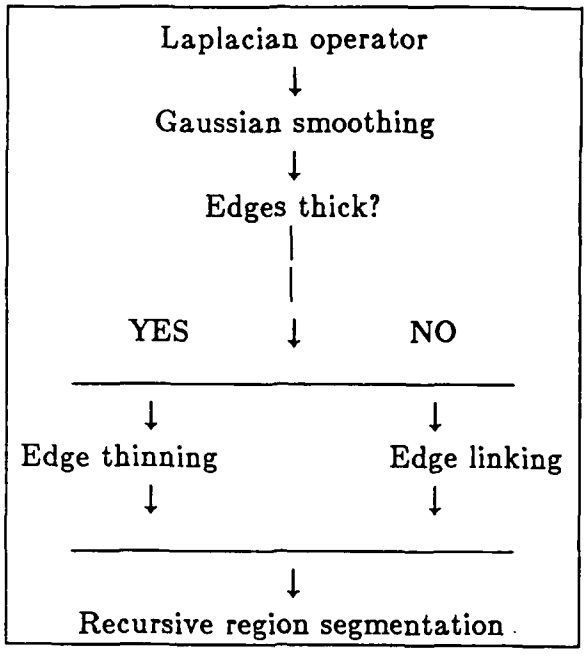

Figure 5.1 Precedence Graph for the Region Splitting Task

Table 5.2 KB Algorithm Classification Parameters

Algorithm mode (SIMD or MIMD)
Number of PE's required
Expected execution time
Input data format
Output data format
Input data autonomy
Output data autonomy

CHAPTER 6
PROPOSED WORK

The previous chapters have described the basis for the proposed work. As a summary of the ideas, current image processing tasks are extremely time and resource intensive. Even parallel processors can have a hard time keeping up with the demand of real-time processing. This research investigates the use of intelligence in the form of an expert system to minimize the execution time of image processing tasks by determining optimal schedules and configurations.

6.1. Proposed Expert System

The proposed expert system will be a part of the intelligent operating system of the parallel processor. The ES will be set up to work with the constraints of the PASM system (i.e., number of processing elements available, possible configurations, reconfiguration overhead, etc.), but changing these limitations will only entail modifying some facts in the ES. In other words, the knowledge of the particular parallel processing system will not be deeply compiled into the expert system.

The language chosen for the expert system is the CLIPS language which was discussed in chapter 4. An example of a CLIPS program is given in appendix A. Parts of the total system will be written in the C language to facilitate quick numeric processing and to ease the operating system interface. For instance, user interfaces will be written in C to allow standard file I/O and fast response time.

The expert system will incorporate rules based on heuristics for its scheduling decisions. Again, the ES will be constructed to be as general as possible concerning the routines it is trying to schedule. The data in the knowledge base will be kept in a file that can be loaded each time the ES is used. This method eliminates the need to recompile the expert system each time a new routine is added to the KB.

The expert system actually has two separate but related functions. The first function is to generate an initial schedule and configuration based on the pre-runtime information available. The second function is to periodically check the execution status so that larger and more accurate pieces of the task can be scheduled. When the ES first makes up the schedule, it will only be able to work with the routines that are between branching points and unclassified routines in the algorithm. As the ES obtains more information such as the resolution of branching decisions or the execution characteristics of unclassified routines, it will be able to fit together or totally restructure the pieces of the schedule it had previously determined.

The user-supplied precedence graph for the current task can in general be in the form of nested IF-THEN-ELSE structures. There is also the need for some kind of looping mechanism. Loops can be accomplished by a WHILE structure similar to those of the C language. For example, the statement "While the region area is greater than 100 pixels, run routine A, then routine B, and, if the texture value is less than 0.6, run routine C" could be entered as follows:

```
while (REGION_AREA > 100)
 {
     ROUTINE_A
     ROUTINE_B
     if (TEXTURE_VALUE < 0.6)
          then {ROUTINE_C}
 }
```

The main emphasis of the research into the expert system will be determining and applying the heuristics to schedule examples like the one above.

The knowledge base that the ES will use can be gradually built up and expanded as new routines are written to fill gaps in the processing capabilities of the KB routines and as routines already in the KB are rewritten in a new form to allow for more compact scheduling.

6.2. Testing the Expert System

There are several issues involved in the testing of the expert system. Obviously, a thorough test of all possible cases that could confront the ES would be the ideal scenario. The criteria for a good testing of the ES are

thoroughness of the test cases, a complete exercising of the heuristics used in the ES, and computational feasibility.

The actual testing can probably best be accomplished on the UNIX operating system where the expert system is being developed. By testing in this manner, we will be able to simulate any desired situation. Artificial data can be supplied to the ES to simulate branching point results, execution times, and input images. Extreme or degenerate cases can be tested to study the results. From cases such as these, new heuristics and modifications of old ones can be determined. In the final stages, testing using actual tasks and data will provide a demonstration of the system. Subject to the status of the PASM hardware and software, demonstration of the execution on PASM may be possible.

6.3. Proposed Test System

Part of the actual testing of the expert system will be the scheduling of a complex image processing task. The use of a real task will give some indication of the results from a non-contrived example.

The image processing task that will be used as part of the testing process will be Yuichi Ohta's "Knowledge-based Interpretation of Outdoor Natural Color Scenes" [Ohta85]. This paper presents an algorithm to produce a consistent interpretation of an input image based on low- and mid-level processing results and knowledge contained in an expert system. His system was chosen for several reasons. As previously mentioned, this task has actually been implemented and will provide a good test case. It was noted in his paper that his algorithm takes a large amount of time to complete (although no specific times were given), so this algorithm is the type that will benefit the most from any speedups. The algorithm also contains branching points and a broad variety of image processing routines which will give a good cross-section of possible processing scenarios.

The algorithm presented in Ohta's paper operates on a 256 by 256 color image and is as follows:

1) Extract textural regions with a 3 by 3 Laplacian window.
2) IF all the regions are textural, THEN proceed to the recognition section.
3) WHILE some region has an area > 50

3a) Compute the histogram of the region.

3b) IF the histogram is monomodal, THEN

3b.1) IF the area of the region ≤ 1536, THEN the processing for this region is done.

3b.2) Recompute the histogram with a window scanning method to try to get more peaks in the histogram.

3b.3) IF the histogram is still monomodal, THEN the processing for this region is done.

3c) ELSE (the histogram has multiple peaks)

3c.1) Select cut-off values and split the region into multiple new regions.

3d) IF the region area is ≤ 8, THEN ignore it because it is too small.

4) Generate a symbolic description of the segmentation results using vertices, holes, boundary segments, etc.

5) Tentatively merge all small regions into one of the large ones.

6) Use an expert system to generate a plan and determine a consistent labeling of the regions.

The final test system will use an expert system written in CLIPS to implement the consistent labeler. CLIPS will be used since the inference engine is available on Purdue's ECN and easy changes can therefore be made to tailor the recognizer for our purposes.

6.4. Expert System Design Methodology

The development of an expert system differs significantly from the development of a non-AI program. A non-AI program (the 9 by 9 Gaussian smoothing of an image, for example) usually has a very specific task to accomplish and the primitives of the application language are both highly formalized and well understood. The steps involved in the task can usually be written out in sufficient detail so that the coding of the task is a relatively straightforward process. Once the code is written, its correctness can then be verified. At this point in the program's development, further changes in the code are only for cosmetic reasons (faster execution time, smaller memory usage, increased user friendliness, etc.) since the program is correctly functioning in its intended purpose.

An expert system, by comparison, generally has a much more nebulous genealogy. Firstly, the task is usually specified in much more general terms such as "schedule image processing tasks on a partitionable parallel processor", "provide a transcription of input speech", or "solve the given symbolic calculus and algebra problems". Although the syntax of the expert system shell may be well defined, the inference engine's firing of the rules generally proceeds in a manner independent of the ordering of those rules. For example, if two rules are ready to be fired, the choice of which one actually does fire might depend on the time since each one was last fired, the number of times each one had previously fired, or the number of conditions on the left hand side of the rule. Since an expert system is largely composed of heuristics about a task domain instead of the details of solving a specific task, coding the rules is neither a straightforward or completely verifiable task. The ability to make decisions under uncertainty, often resulting in a probability that the recommended action will succeed (i.e. a confidence value) opposes the idea of an exacting verification process.

The design of an expert system, therefore, is an iterative refinement process. In the case of the expert system proposed in the work, the refinement will be towards shorter execution times for the image processing tasks that are being scheduled. The first phase of the research consists of two parts. The first part will be the acquisition of the knowledge base. For the routines that presently exist, their performance characteristics (as specified in chapter 5) will be tabulated. Necessary routines which have not yet been written will have expected characteristics tabulated. Since the structure of the expert system will not depend on the knowledge base, exact values for the algorithm characteristics, such as the number of PE's required, is not necessary for the expert system design or testing. However, once the expert system is running in its intended capacity, the usefullness of its decisions will depend heavily on the accuracy of the information contained in the knowledge base.

The second part of the initial phase is to build a small expert system to validate the design concept. On the order of fifty rules will be used to implement a simplified statement of the proposed problem, specifically: "Given the current configuration and the next routine to be run, which knowledge base version of that routine should be used?". Once this basic expert system is functional in the sense of giving acceptable advice, the work can proceed to phase two, the refinement process.

The second phase has two goals. First, the expert system will be extended to deal with the full problem statement. Rules will be added so that the expert system can look ahead to the routines waiting to be done, deal with branching points and iterations in the specified task, deal with uncertainties about routines not in the knowledge base, etc. The second goal will be to refine the rules of the expert system to achieve decreased execution time for the image processing tasks, achieve decreased execution time within the expert system itself, and to have increased robustness in handling extreme or degenerate cases.

The refinement process, and in part the building of the initial expert system, will involve repeated testing with more and more complex tasks. New rules and changes to existing rules will come from both positive and negative results of the test cases. In this manner, instances where inappropriate decisions are made can be reduced and better scheduling can be achieved.

6.5. Conclusion

This paper has proposed an intelligent scheduler in the form of an expert system. The goal will be to accomplish improved scheduling of image processing algorithms on a partitionable parallel processing system. Once this goal is achieved, it will provide an aid in the production of image processing tasks that require real-time operation. One of the advantages of the completed expert system is that it will act as a buffer between the user and the system details and therefore will not require the user to have expert knowledge in either parallel algorithms or the target parallel processor.

Contributions from this work will be in the areas of image processing, artificial intelligence, and operating system design for partitionable parallel processors. The image processing contribution will come from the ability to write a general algorithm specification that is independent of the parallel processor being used and the number of different versions of a routine in the knowledge base. This ability will therefore aid in the design of portable image processing code since the same algorithm will be able to be run on any given partitionable parallel processor that is using the scheduling expert system. The artificial intelligence contribution will come in the form of the scheduling expert system itself. The design of the expert system will include research into the area of decision under uncertainty. The contribution in the design of operating systems for partitionable parallel processors will be the availability of a general scheduling system for image processing tasks. The

expert system's rules will be applicable to any partitionable parallel processor. Only the facts will have to be changed to account for hardware differences between systems.

LIST OF REFERENCES

[Adam82] G. B. Adams III and H. J. Siegel, "The Extra Stage Cube: a Fault-Tolerant Interconnection Network for Supersystems," *IEEE Transactions on Computers,* Vol. C-31, May 1982, pp. 443-454

[Delp85] E. J. Delp, H. J. Siegel, A. Whinston, and L. H. Jamieson, "An Intelligent Operating System for Executing Image Understanding Tasks on a Reconfigurable Parallel Architecture," *Proceedings of the IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management,* Nov. 1985, pp. 217-224

[Duff82] M. J. B. Duff, "Parallel Algorithms and Their Influence on the Specification of Application Problems," in *Multicomputers and Image Processing: Algorithms and Programs,* K. Preston and L. Uhr, eds., Academic Press, New York, NY, 1982, pp. 261-274

[Erma80] L. D. Erman et al., "Hearsay-II Speech-Understanding System: Integrating Knowledge To Resolve Uncertainty," *Computing Surveys,* Vol. 12, No. 2, Feb. 1980, pp. 213-253.

[Flyn66] M. J. Flynn, "Very High-Speed Computing Systems," *Proc. IEEE,* Vol. 54, Dec. 1966, pp. 1901-1909

[Huec71] M. H. Hueckel, "An Operator Which Locates Edges in Digitized Pictures," *Journal of the ACM,* Vol. 18, 1971, pp. 113-125.

[Huec73] M. H. Hueckel, "A Local Visual Edge Operator Which Recognizes Edges and Lines," *Journal of the ACM,* Vol. 20, 1973, pp. 634-647

[Jami85] L. J. Jamieson, "The Mapping of Parallel Algorithms To Reconfigurable Parallel Architectures," in *Languages, Architectures, and Algorithms for Parallel Image Processing,* M. J. B. Duff, ed., Academic Press, 1985.

[Kryg76] A. J. Krygiel, "An Implementation of the Hadamard Transform on the STARAN Associative Array Processors," *1976*

International Con. on Parallel Processing, Aug. 1976, p. 34

[Kueh85] J. T. Kuehn, T. Schwederski, and H. J. Siegel, "Design of a 1024-Processor PASM System," *Proceedings of the First International Conference on Supercomputing Systems,* Dec. 1985

[Lind80] R. K. Lindsay et al., *Applications of Artificial Intelligence for Organic Chemistry: The Dendral Project,* McGraw-Hill, New York, 1980

[Lope86] F. M. Lopez *CLIPS Reference Manual* Artificial Intelligence Section, Mission Planning and Analysis Division, NASA, July 1986

[Mart71] W. A. Martin and R. J. Fateman, "The Macsyma System," *Proc. Second Symp. Symbolic and Algebraic Manipulation,* 1971, pp. 59-75.

[Ohta85] Y. Ohta, *Knowledge-based Interpretation of Outdoor Natural Color Scenes,* Pitman Publishing, Massachusetts, 1985

[Prew70] J. M. S. Prewitt, "Object Enhancement and Extraction," in *Picture Processing and Psychopictorics,* ed. B. S. Lipkin and A. Rosenfeld, Academic Press, New York, 1970, pp. 75-149

[Rice85] T. A. Rice and L. J. Siegel "Parallel Processing for Computer Vision," in *Parallel Integrated Technology for Image Processing,* S. Levialdi ed., Academic Press, 1985

[Robe65] L. G. Roberts, "Machine Perception of Three-Dimensional Solids," in *Optical and Electro-Optical Information Processing,* ed. J. T. Tippet et al, MIT Press, Cambridge, Mass., 1965, pp. 159-197.

[Rohr77] D. Rohrbacker and J. L. Potter, "Image Processing with the STARAN Parallel Computer," *Computer,* Vol. 10, No. 8, Aug. 1977, pp. 54-59

[Schw86] T. Schwederski, "The PASM Parallel Processing System: Hardware Design and Intelligent Operating System Concepts," Doctoral Thesis Proposal, 1986.

[Shor76] E. H. Shortliffe, *Computer-Based Medical Consultation: MYCIN,* American Elsevier, New York, 1976.

[Sieg81] H. J. Siegel, L. J. Siegel, F. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: a Partitionable

Multimicrocomputer SIMD/MIMD System for Image Processing and Pattern Recognition," *IEEE Transactions on Computers,* Vol. C-30, no. 12, December 1981, pp. 934-947.

[Sieg86] H. J. Siegel, T. Schwederski, J. T. Kuehn, and N. J. Davis IV, "An Overview of the PASM Parallel Processing System," in *Tutorial on Computer Architecture,* D. D. Gajski, V. M. Milutinovic, H. J. Siegel, and B. P. Furht, eds., IEEE Computer Society Press, Washington, D. C., to appear 1986

APPENDIX A

The following code is an example of a program written in the CLIPS expert system language. It implements the "monkey and banana" problem. Example output is shown for the case where no diagnostics are requested and the case where all possible diagnostics are requested.

```
;Created at Nasa's
;Mission Planning & Analysis Division's
;Artificial Intelligence Section
;Johnson Space Center, Houston Texas, 77058
;
;This is the monkey and the ladder problem.
;Created by Chris Culbert for The Lisp Machine, 3600, using ART.
;Modified from ART to CLIPS by Frank Lopez.
;
;

(defrule GI-MOVE-LADDER-OBJECT
      (goal-is-to active holds ?obj)
      (object ?obj ?place ? ceiling)
  =>
      (assert (goal-is-to active move ladder ?place)))

(defrule GI-CLIMB-LADDER
      (goal-is-to active holds ?obj)
      (object ?obj ?place ? ceiling)
      (object ladder ?place ? ?)
  =>
      (assert (goal-is-to active on ladder)))
```

```
(defrule GET-OBJECT
      ?f1<-(goal-is-to active holds ?obj)
      (object ?obj ?place ? ceiling)
      (object ladder ?place ? ?)
      ?f2<-(monkey ?place ladder blank)
 =>
      (printout (monkey grabs the ?obj))
      (retract (?f1))
      (retract (?f2))
      (assert (goal-is-to satisfied holds ?obj))
      (assert (monkey ?place ladder ?obj)))


(defrule GI-WALK-ON-FLOOR
      (goal-is-to active holds ?obj)
      (object ?obj ?place ? floor)
 =>
      (assert (goal-is-to active walk-to ?place)))


(defrule GI-DROP-OBJECT
      #(salience -10)
      (goal-is-to satisfied holds ?obj)
      (monkey ?place ? ?obj)
 =>
      (assert (goal-is-to active holds blank)))


(defrule GRAB-OBJECT-ON-FLOOR
      ?f1<-(goal-is-to active holds ?obj)
      (object ?obj ?place ? floor)
      ?f2<-(monkey ?place ?on blank)
 =>
      (printout (monkey grabs the ?obj))
      (retract (?f1))
      (retract (?f2))
      (assert (goal-is-to satisfied holds ?obj))
      (assert (monkey ?place ?on ?obj)))
```

```
(defrule GI-HOLD-LIGHT-OBJECT
     (goal-is-to active move ?obj ?place)
     (object ?obj ~?place light ?)
  =>
     (assert (goal-is-to active holds ?obj)))


(defrule GI-WALK-TO-LT-OBJECT
     (goal-is-to active move ?obj ?place)
     (object ?obj ~?place light ?)
     (monkey ? ? ?obj)
  =>
     (assert (goal-is-to active walk-to ?place)))


(defrule OBJECT-MOVED-TO-LOC
     ?f1<-(goal-is-to active move ?obj ?place)
     (object ?obj ?place light ?)
  =>
     (retract (?f1))
     (assert (goal-is-to satisfied move ?obj ?place)))


(defrule GI-GET-ON-FLOOR
     (goal-is-to active walk-to ?place)
     (monkey ~?place ~floor ?)
  =>
     (assert (goal-is-to active on floor)))

(defrule WALK-OBJ-EMP-HANDED
     ?f1<-(goal-is-to active walk-to ?place)
     ?f2<-(monkey ?c&~?place floor blank)
  =>
     (printout (monkey walks to ?place))
     (retract (?f1))
     (retract (?f2))
```

```
      (assert (goal-is-to satisfied walk-to ?place))
      (assert (monkey ?place floor blank)))


(defrule WALK-OBJ-W-OTHER-OBJ
      ?f1<-(goal-is-to active walk-to ?place)
      ?f2<-(monkey ?c&~?place floor ?obj&~blank)
      ?f3<-(object ?obj ?at ?mass ?on)
  =>
      (printout (monkey walks to ?place holding ?obj))
      (retract (?f1))
      (retract (?f2))
      (retract (?f3))
      (assert (goal-is-to satisfied walk-to ?place))
      (assert (monkey ?place floor ?obj))
      (assert (object ?obj ?place ?mass ?on)))


(defrule jump-onto-floor
      ?f1<-(goal-is-to active on floor)
      ?f2<-(monkey ?at ?on&~floor ?obj)
  =>
      (printout (monkey jumps onto the floor))
      (retract (?f1))
      (retract (?f2))
      (assert (goal-is-to satisfied on floor))
      (assert (monkey ?at floor ?obj)))


(defrule gi-hold-obj-your-on
      (goal-is-to active on ?obj)
      (object ?obj ?place ? ?)
  =>
      (assert (goal-is-to active walk-to ?place)))


(defrule gi-drop-desired-obj
      (goal-is-to active on ?obj)
```

```
        (object ?obj ?place ? ?)
        (monkey ?place ? ?)
    =>
        (assert (goal-is-to active holds blank)))


(defrule climb-onto-obj
        ?f1<-(goal-is-to active on ?obj)
        (object ?obj ?place ? ?)
        ?f3<-(monkey ?place ?on blank)
    =>
        (printout (Monkey climbs onto ?obj))
        (retract (?f1))
        (retract (?f3))
        (assert (goal-is-to satisfied on ?obj))
        (assert (monkey ?place ?obj blank)))


(defrule drop-object
        ?f1<-(goal-is-to active holds blank)
        ?f2<-(monkey ?place ?on ?x&~blank)
    =>
        (printout (monkey drops ?x))
        (retract (?f1))
        (retract (?f2))
        (assert (goal-is-to satisfied holds blank))
        (assert (monkey ?place ?on blank)))


(defrule gi-not-processed
        #(salience -100)
        ?f1<-(goal-is-to active $?rest)
    =>
        (retract (?f1))
        (assert (goal-is-to not-processed $?rest)))


(defrule startup
```

```
    (initial-fact)
 =>
    (assert (monkey t5-7 couch blank))
    (assert (object couch t5-7 heavy floor))
    (assert (object bananas t2-2 light ceiling))
    (assert (object ladder t9-5 light floor))
    (assert (goal-is-to active holds bananas)))

(defrule satisfy-hunger
    ?f1<-(monkey ?place ?on bananas)
 =>
    (printout (Monkey Eats the Bananas))
    (retract (?f1))
    (assert (monkey ?place ?on blank)))
```

The following questions are printed at the beginning of the run of the expert system. The user's responses immediately follow the "?" of the question. The results of the run are printed after the "Watch is off." message.

```
          CLIPS Version 1.1  May 29, 1985: FML
-----------------------------------------
Do you wish to step through the execution (yes/no) ?n
Stepper is off
Do you wish to have the agenda & fact list displayed (yes/no) ?n
AGENDA & Fact list will not be displayed
Do you wish to have the watch mode on (yes/no) ?n
Watch is off.
monkey jumps onto the floor
monkey walks to t9-5
monkey grabs the ladder
monkey walks to t2-2 holding ladder
monkey drops ladder
Monkey climbs onto ladder
monkey grabs the bananas
Monkey Eats the Bananas
Execution completed successfully
```

The following run of the expert system is the same as the previous one except that all possible tracing modes are turned on.

```
          CLIPS Version 1.1  May 29, 1985: FML
---------------------------------------
Do you wish to step through the execution (yes/no) ?y
Stepper is on
Do you wish to have the agenda & fact list displayed (yes/no) ?y
AGENDA & Fact list will be displayed
Do you wish to have the watch mode on (yes/no) ?y
Watch is on.
Compiling GI-MOVE-LADDER-OBJECT
Compiling GI-CLIMB-LADDER
Compiling GET-OBJECT
Compiling GI-WALK-ON-FLOOR
Compiling GI-DROP-OBJECT
Compiling GRAB-OBJECT-ON-FLOOR
Compiling GI-HOLD-LIGHT-OBJECT
Compiling GI-WALK-TO-LT-OBJECT
Compiling OBJECT-MOVED-TO-LOC
Compiling GI-GET-ON-FLOOR
Compiling WALK-OBJ-EMP-HANDED
Compiling WALK-OBJ-W-OTHER-OBJ
Compiling jump-onto-floor
Compiling gi-hold-obj-your-on
Compiling gi-drop-desired-obj
Compiling climb-onto-obj
Compiling drop-object
Compiling gi-not-processed
Compiling startup
Compiling satisfy-hunger
==> Activation    0 startup:  f-0 .


      FACTS IN THE DATABASE
**********************************
```

```
id: 0; order: 0 (initial-fact)
**********************************
******************AGENDA**********************
ID: 0 ORDER: 0   startup is ready to fire   0.00
**********************************************

startup had fired
FIRE    0            startup:  f-0 .
==> Fact f-1: (monkey t5-7 couch blank)
==> Fact f-2: (object couch t5-7 heavy floor)
==> Fact f-3: (object bananas t2-2 light ceiling)
==> Fact f-4: (object ladder t9-5 light floor)
==> Fact f-5: (goal-is-to active holds bananas)
==> Activation    1 GI-MOVE-LADDER-OBJECT:  f-3  f-5 .
==> Activation    2 gi-not-processed:  f-5 .


     FACTS IN THE DATABASE
**********************************
id: 0; order: 0 (initial-fact)
id: 1; order: 1 (monkey t5-7 couch blank)
id: 2; order: 2 (object couch t5-7 heavy floor)
id: 3; order: 3 (object bananas t2-2 light ceiling)
id: 4; order: 4 (object ladder t9-5 light floor)
id: 5; order: 5 (goal-is-to active holds bananas)
**********************************
******************AGENDA**********************
ID: 1 ORDER: 0   GI-MOVE-LADDER-OBJECT is ready to fire   0.00
ID: 2 ORDER: 1   gi-not-processed is ready to fire -100.00
**********************************************

GI-MOVE-LADDER-OBJECT had fired
FIRE    1 GI-MOVE-LADDER-OBJECT:  f-3  f-5 .
==> Fact f-6: (goal-is-to active move ladder t2-2)
==> Activation    3 GI-HOLD-LIGHT-OBJECT:  f-4  f-6 .
==> Activation    4 gi-not-processed:  f-6 .
```

FACTS IN THE DATABASE

****

id: 0; order: 0 (initial-fact)
id: 1; order: 1 (monkey t5-7 couch blank)
id: 2; order: 2 (object couch t5-7 heavy floor)
id: 3; order: 3 (object bananas t2-2 light ceiling)
id: 4; order: 4 (object ladder t9-5 light floor)
id: 5; order: 5 (goal-is-to active holds bananas)
id: 6; order: 6 (goal-is-to active move ladder t2-2)

****

**********AGENDA**********

ID: 3 ORDER: 0 GI-HOLD-LIGHT-OBJECT is ready to fire 0.00
ID: 4 ORDER: 1 gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 2 gi-not-processed is ready to fire -100.00

****

GI-HOLD-LIGHT-OBJECT had fired
FIRE 3 GI-HOLD-LIGHT-OBJECT: f-4 f-6 .
==> Fact f-7: (goal-is-to active holds ladder)
==> Activation 5 GI-WALK-ON-FLOOR: f-4 f-7 .
==> Activation 6 gi-not-processed: f-7 .

FACTS IN THE DATABASE

****

id: 0; order: 0 (initial-fact)
id: 1; order: 1 (monkey t5-7 couch blank)
id: 2; order: 2 (object couch t5-7 heavy floor)
id: 3; order: 3 (object bananas t2-2 light ceiling)
id: 4; order: 4 (object ladder t9-5 light floor)
id: 5; order: 5 (goal-is-to active holds bananas)
id: 6; order: 6 (goal-is-to active move ladder t2-2)
id: 7; order: 7 (goal-is-to active holds ladder)

****

**********AGENDA**********

ID: 5 ORDER: 0 GI-WALK-ON-FLOOR is ready to fire 0.00
ID: 6 ORDER: 1 gi-not-processed is ready to fire -100.00

```
ID: 4 ORDER: 2   gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 3   gi-not-processed is ready to fire -100.00
*****************************************

GI-WALK-ON-FLOOR had fired
FIRE   5    GI-WALK-ON-FLOOR:  f-4  f-7 .
==> Fact f-8: (goal-is-to active walk-to t9-5)
==> Activation    7 GI-GET-ON-FLOOR:  f-1  f-8 .
==> Activation    8 gi-not-processed:  f-8 .


     FACTS IN THE DATABASE
*********************************
id: 0; order: 0 (initial-fact)
id: 1; order: 1 (monkey t5-7 couch blank)
id: 2; order: 2 (object couch t5-7 heavy floor)
id: 3; order: 3 (object bananas t2-2 light ceiling)
id: 4; order: 4 (object ladder t9-5 light floor)
id: 5; order: 5 (goal-is-to active holds bananas)
id: 6; order: 6 (goal-is-to active move ladder t2-2)
id: 7; order: 7 (goal-is-to active holds ladder)
id: 8; order: 8 (goal-is-to active walk-to t9-5)
*********************************
******************AGENDA*********************
ID: 7 ORDER: 0   GI-GET-ON-FLOOR is ready to fire   0.00
ID: 8 ORDER: 1   gi-not-processed is ready to fire -100.00
ID: 6 ORDER: 2   gi-not-processed is ready to fire -100.00
ID: 4 ORDER: 3   gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 4   gi-not-processed is ready to fire -100.00
*********************************************

GI-GET-ON-FLOOR had fired
FIRE   7    GI-GET-ON-FLOOR:  f-1  f-8 .
==> Fact f-9: (goal-is-to active on floor)
==> Activation    9 jump-onto-floor:  f-1  f-9 .
==> Activation   10 gi-not-processed:  f-9 .
```

```
     FACTS IN THE DATABASE
*********************************
id: 0; order: 0 (initial-fact)
id: 1; order: 1 (monkey t5-7 couch blank)
id: 2; order: 2 (object couch t5-7 heavy floor)
id: 3; order: 3 (object bananas t2-2 light ceiling)
id: 4; order: 4 (object ladder t9-5 light floor)
id: 5; order: 5 (goal-is-to active holds bananas)
id: 6; order: 6 (goal-is-to active move ladder t2-2)
id: 7; order: 7 (goal-is-to active holds ladder)
id: 8; order: 8 (goal-is-to active walk-to t9-5)
id: 9; order: 9 (goal-is-to active on floor)
*********************************
******************AGENDA*******************
ID: 9 ORDER: 0   jump-onto-floor is ready to fire   0.00
ID: 10 ORDER: 1   gi-not-processed is ready to fire -100.00
ID: 8 ORDER: 2   gi-not-processed is ready to fire -100.00
ID: 6 ORDER: 3   gi-not-processed is ready to fire -100.00
ID: 4 ORDER: 4   gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 5   gi-not-processed is ready to fire -100.00
********************************************

jump-onto-floor had fired
FIRE   9      jump-onto-floor:  f-1  f-9 .
monkey jumps onto the floor
<== Activation   9 jump-onto-floor:  f-1  f-9 .
<== Activation   10 gi-not-processed:  f-9 .
<== Fact f-9: (goal-is-to active on floor)
<== Fact f-1: (monkey t5-7 couch blank)
==> Fact f-10: (goal-is-to satisfied on floor)
==> Fact f-11: (monkey t5-7 floor blank)
==> Activation   11 WALK-OBJ-EMP-HANDED:  f-8  f-11 .


     FACTS IN THE DATABASE
*********************************
id: 0; order: 0 (initial-fact)
```

id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 4; order: 3 (object ladder t9-5 light floor)
id: 5; order: 4 (goal-is-to active holds bananas)
id: 6; order: 5 (goal-is-to active move ladder t2-2)
id: 7; order: 6 (goal-is-to active holds ladder)
id: 8; order: 7 (goal-is-to active walk-to t9-5)
id: 10; order: 8 (goal-is-to satisfied on floor)
id: 11; order: 9 (monkey t5-7 floor blank)
****
**********AGENDA**********
ID: 11 ORDER: 0 WALK-OBJ-EMP-HANDED is ready to fire 0.00
ID: 8 ORDER: 1 gi-not-processed is ready to fire -100.00
ID: 6 ORDER: 2 gi-not-processed is ready to fire -100.00
ID: 4 ORDER: 3 gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 4 gi-not-processed is ready to fire -100.00
****

WALK-OBJ-EMP-HANDED had fired
FIRE 11 WALK-OBJ-EMP-HANDED: f-8 f-11 .
monkey walks to t9-5
<== Activation 11 WALK-OBJ-EMP-HANDED: f-8 f-11 .
<== Activation 8 gi-not-processed: f-8 .
<== Fact f-8: (goal-is-to active walk-to t9-5)
<== Fact f-11: (monkey t5-7 floor blank)
==> Fact f-12: (goal-is-to satisfied walk-to t9-5)
==> Fact f-13: (monkey t9-5 floor blank)
==> Activation 12 GRAB-OBJECT-ON-FLOOR: f-4 f-7 f-13 .

FACTS IN THE DATABASE
****
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 4; order: 3 (object ladder t9-5 light floor)
id: 5; order: 4 (goal-is-to active holds bananas)

id: 6; order: 5 (goal-is-to active move ladder t2-2)
id: 7; order: 6 (goal-is-to active holds ladder)
id: 10; order: 7 (goal-is-to satisfied on floor)
id: 12; order: 8 (goal-is-to satisfied walk-to t9-5)
id: 13; order: 9 (monkey t9-5 floor blank)
****
**********AGENDA**********
ID: 12 ORDER: 0 GRAB-OBJECT-ON-FLOOR is ready to fire 0.00
ID: 6 ORDER: 1 gi-not-processed is ready to fire -100.00
ID: 4 ORDER: 2 gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 3 gi-not-processed is ready to fire -100.00
****

GRAB-OBJECT-ON-FLOOR had fired
FIRE 12 GRAB-OBJECT-ON-FLOOR: f-4 f-7 f-13 .
monkey grabs the ladder
<== Activation 12 GRAB-OBJECT-ON-FLOOR: f-4 f-7 f-13 .
<== Activation 6 gi-not-processed: f-7 .
<== Fact f-7: (goal-is-to active holds ladder)
<== Fact f-13: (monkey t9-5 floor blank)
==> Fact f-14: (goal-is-to satisfied holds ladder)
==> Fact f-15: (monkey t9-5 floor ladder)
==> Activation 13 GI-DROP-OBJECT: f-14 f-15 .
==> Activation 14 GI-WALK-TO-LT-OBJECT: f-4 f-6 f-15 .

FACTS IN THE DATABASE
****
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 4; order: 3 (object ladder t9-5 light floor)
id: 5; order: 4 (goal-is-to active holds bananas)
id: 6; order: 5 (goal-is-to active move ladder t2-2)
id: 10; order: 6 (goal-is-to satisfied on floor)
id: 12; order: 7 (goal-is-to satisfied walk-to t9-5)
id: 14; order: 8 (goal-is-to satisfied holds ladder)

```
id: 15; order: 9 (monkey t9-5 floor ladder)
**********************************
******************AGENDA*******************
ID: 14 ORDER: 0   GI-WALK-TO-LT-OBJECT is ready to fire   0.00
ID: 13 ORDER: 1   GI-DROP-OBJECT is ready to fire -10.00
ID: 4 ORDER: 2   gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 3   gi-not-processed is ready to fire -100.00
*********************************************

GI-WALK-TO-LT-OBJECT had fired
FIRE   14 GI-WALK-TO-LT-OBJECT:  f-4  f-6  f-15 .
==> Fact f-16: (goal-is-to active walk-to t2-2)
==> Activation   15 WALK-OBJ-W-OTHER-OBJ:  f-4  f-15  f-16 .
==> Activation   16 gi-not-processed:  f-16 .


     FACTS IN THE DATABASE
**********************************
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 4; order: 3 (object ladder t9-5 light floor)
id: 5; order: 4 (goal-is-to active holds bananas)
id: 6; order: 5 (goal-is-to active move ladder t2-2)
id: 10; order: 6 (goal-is-to satisfied on floor)
id: 12; order: 7 (goal-is-to satisfied walk-to t9-5)
id: 14; order: 8 (goal-is-to satisfied holds ladder)
id: 15; order: 9 (monkey t9-5 floor ladder)
id: 16; order: 10 (goal-is-to active walk-to t2-2)
**********************************
******************AGENDA*******************
ID: 15 ORDER: 0   WALK-OBJ-W-OTHER-OBJ is ready to fire   0.00
ID: 13 ORDER: 1   GI-DROP-OBJECT is ready to fire -10.00
ID: 16 ORDER: 2   gi-not-processed is ready to fire -100.00
ID: 4 ORDER: 3   gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 4   gi-not-processed is ready to fire -100.00
*********************************************
```

WALK-OBJ-W-OTHER-OBJ had fired
FIRE 15 WALK-OBJ-W-OTHER-OBJ: f-4 f-15 f-16 .
monkey walks to t2-2 holding ladder
<== Activation 15 WALK-OBJ-W-OTHER-OBJ: f-4 f-15 f-16 .
<== Activation 16 gi-not-processed: f-16 .
<== Fact f-16: (goal-is-to active walk-to t2-2)
<== Activation 13 GI-DROP-OBJECT: f-14 f-15 .
<== Fact f-15: monkey t9-5 floor ladder)
<== Fact f-4: (object ladder t9-5 light floor)
==> Fact f-17: (goal-is-to satisfied walk-to t2-2)
==> Fact f-18: (monkey t2-2 floor ladder)
==> Fact f-19: (object ladder t2-2 light floor)
==> Activation 17 GI-DROP-OBJECT: f-14 f-18 .
==> Activation 18 GI-CLIMB-LADDER: f-3 f-5 f-19 .
==> Activation 19 OBJECT-MOVED-TO-LOC: f-6 f-19 .

FACTS IN THE DATABASE
****
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 5; order: 3 (goal-is-to active holds bananas)
id: 6; order: 4 (goal-is-to active move ladder t2-2)
id: 10; order: 5 (goal-is-to satisfied on floor)
id: 12; order: 6 (goal-is-to satisfied walk-to t9-5)
id: 14; order: 7 (goal-is-to satisfied holds ladder)
id: 17; order: 8 (goal-is-to satisfied walk-to t2-2)
id: 18; order: 9 (monkey t2-2 floor ladder)
id: 19; order: 10 (object ladder t2-2 light floor)
****
**********AGENDA**********
ID: 19 ORDER: 0 OBJECT-MOVED-TO-LOC is ready to fire 0.00
ID: 18 ORDER: 1 GI-CLIMB-LADDER is ready to fire 0.00
ID: 17 ORDER: 2 GI-DROP-OBJECT is ready to fire -10.00
ID: 4 ORDER: 3 gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 4 gi-not-processed is ready to fire -100.00

```
*********************************************

OBJECT-MOVED-TO-LOC had fired
FIRE  19  OBJECT-MOVED-TO-LOC:  f-6  f-19 .
<== Activation  19 OBJECT-MOVED-TO-LOC:  f-6  f-19 .
<== Activation   4 gi-not-processed:  f-6 .
<== Fact f-6: (goal-is-to active move ladder t2-2)
==> Fact f-20: (goal-is-to satisfied move ladder t2-2)



     FACTS IN THE DATABASE
**********************************
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 5; order: 3 (goal-is-to active holds bananas)
id: 10; order: 4 (goal-is-to satisfied on floor)
id: 12; order: 5 (goal-is-to satisfied walk-to t9-5)
id: 14; order: 6 (goal-is-to satisfied holds ladder)
id: 17; order: 7 (goal-is-to satisfied walk-to t2-2)
id: 18; order: 8 (monkey t2-2 floor ladder)
id: 19; order: 9 (object ladder t2-2 light floor)
id: 20; order: 10 (goal-is-to satisfied move ladder t2-2)
**********************************
**********AGENDA**********
ID: 18 ORDER: 0   GI-CLIMB-LADDER is ready to fire   0.00
ID: 17 ORDER: 1   GI-DROP-OBJECT is ready to fire -10.00
ID: 2 ORDER: 2   gi-not-processed is ready to fire -100.00
*********************************************

GI-CLIMB-LADDER had fired
FIRE  18     GI-CLIMB-LADDER:  f-3  f-5  f-19 .
==> Fact f-21: (goal-is-to active on ladder)
==> Activation   20 gi-hold-obj-your-on:  f-19  f-21 .
==> Activation   21 gi-drop-desired-obj:  f-18  f-19  f-21 .
==> Activation   22 gi-not-processed:  f-21 .
```

```
     FACTS IN THE DATABASE
**********************************
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 5; order: 3 (goal-is-to active holds bananas)
id: 10; order: 4 (goal-is-to satisfied on floor)
id: 12; order: 5 (goal-is-to satisfied walk-to t9-5)
id: 14; order: 6 (goal-is-to satisfied holds ladder)
id: 17; order: 7 (goal-is-to satisfied walk-to t2-2)
id: 18; order: 8 (monkey t2-2 floor ladder)
id: 19; order: 9 (object ladder t2-2 light floor)
id: 20; order: 10 (goal-is-to satisfied move ladder t2-2)
id: 21; order: 11 (goal-is-to active on ladder)
**********************************
******************AGENDA**********************
ID: 21 ORDER: 0   gi-drop-desired-obj is ready to fire   0.00
ID: 20 ORDER: 1   gi-hold-obj-your-on is ready to fire   0.00
ID: 17 ORDER: 2   GI-DROP-OBJECT is ready to fire -10.00
ID: 22 ORDER: 3   gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 4   gi-not-processed is ready to fire -100.00
**********************************************

gi-drop-desired-obj had fired
FIRE   21  gi-drop-desired-obj:  f-18  f-19  f-21 .
==> Fact f-22: (goal-is-to active holds blank)
==> Activation   23 drop-object:  f-18  f-22 .
==> Activation   24 gi-not-processed:  f-22 .


     FACTS IN THE DATABASE
**********************************
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 5; order: 3 (goal-is-to active holds bananas)
id: 10; order: 4 (goal-is-to satisfied on floor)
```

```
id: 12; order: 5 (goal-is-to satisfied walk-to t9-5)
id: 14; order: 6 (goal-is-to satisfied holds ladder)
id: 17; order: 7 (goal-is-to satisfied walk-to t2-2)
id: 18; order: 8 (monkey t2-2 floor ladder)
id: 19; order: 9 (object ladder t2-2 light floor)
id: 20; order: 10 (goal-is-to satisfied move ladder t2-2)
id: 21; order: 11 (goal-is-to active on ladder)
id: 22; order: 12 (goal-is-to active holds blank)
**********************************
*******************AGENDA*********************
ID: 23 ORDER: 0   drop-object is ready to fire   0.00
ID: 20 ORDER: 1   gi-hold-obj-your-on is ready to fire   0.00
ID: 17 ORDER: 2   GI-DROP-OBJECT is ready to fire -10.00
ID: 24 ORDER: 3   gi-not-processed is ready to fire -100.00
ID: 22 ORDER: 4   gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 5   gi-not-processed is ready to fire -100.00
**********************************************

drop-object had fired
FIRE  23        drop-object:  f-18  f-22 .
monkey drops ladder
<== Activation   23 drop-object:  f-18  f-22 .
<== Activation   24 gi-not-processed:  f-22 .
<== Fact f-22: (goal-is-to active holds blank)
<== Activation   17 GI-DROP-OBJECT:  f-14  f-18 .
<== Fact f-18: (monkey t2-2 floor ladder)
==> Fact f-23: (goal-is-to satisfied holds blank)
==> Fact f-24: (monkey t2-2 floor blank)
==> Activation   25 GI-DROP-OBJECT:  f-23  f-24 .
==> Activation   26 gi-drop-desired-obj:  f-19  f-21  f-24 .
==> Activation   27 climb-onto-obj:  f-19  f-21  f-24 .


     FACTS IN THE DATABASE
**********************************
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
```

id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 5; order: 3 (goal-is-to active holds bananas)
id: 10; order: 4 (goal-is-to satisfied on floor)
id: 12; order: 5 (goal-is-to satisfied walk-to t9-5)
id: 14; order: 6 (goal-is-to satisfied holds ladder)
id: 17; order: 7 (goal-is-to satisfied walk-to t2-2)
id: 19; order: 8 (object ladder t2-2 light floor)
id: 20; order: 9 (goal-is-to satisfied move ladder t2-2)
id: 21; order: 10 (goal-is-to active on ladder)
id: 23; order: 11 (goal-is-to satisfied holds blank)
id: 24; order: 12 (monkey t2-2 floor blank)
****
**********AGENDA**********
ID: 27 ORDER: 0 climb-onto-obj is ready to fire 0.00
ID: 26 ORDER: 1 gi-drop-desired-obj is ready to fire 0.00
ID: 20 ORDER: 2 gi-hold-obj-your-on is ready to fire 0.00
ID: 25 ORDER: 3 GI-DROP-OBJECT is ready to fire -10.00
ID: 22 ORDER: 4 gi-not-processed is ready to fire -100.00
ID: 2 ORDER: 5 gi-not-processed is ready to fire -100.00
****

climb-onto-obj had fired
FIRE 27 climb-onto-obj: f-19 f-21 f-24 .
Monkey climbs onto ladder
<== Activation 27 climb-onto-obj: f-19 f-21 f-24 .
<== Activation 26 gi-drop-desired-obj: f-19 f-21 f-24 .
<== Activation 20 gi-hold-obj-your-on: f-19 f-21 .
<== Activation 22 gi-not-processed: f-21 .
<== Fact f-21: (goal-is-to active on ladder)
<== Activation 25 GI-DROP-OBJECT: f-23 f-24 .
<== Fact f-24: (monkey t2-2 floor blank)
==> Fact f-25: (goal-is-to satisfied on ladder)
==> Fact f-26: (monkey t2-2 ladder blank)
==> Activation 28 GET-OBJECT: f-3 f-5 f-19 f-26 .
==> Activation 29 GI-DROP-OBJECT: f-23 f-26 .

```
     FACTS IN THE DATABASE
**********************************
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 5; order: 3 (goal-is-to active holds bananas)
id: 10; order: 4 (goal-is-to satisfied on floor)
id: 12; order: 5 (goal-is-to satisfied walk-to t9-5)
id: 14; order: 6 (goal-is-to satisfied holds ladder)
id: 17; order: 7 (goal-is-to satisfied walk-to t2-2)
id: 19; order: 8 (object ladder t2-2 light floor)
id: 20; order: 9 (goal-is-to satisfied move ladder t2-2)
id: 23; order: 10 (goal-is-to satisfied holds blank)
id: 25; order: 11 (goal-is-to satisfied on ladder)
id: 26; order: 12 (monkey t2-2 ladder blank)
**********************************
*******************AGENDA*********************
ID: 28 ORDER: 0   GET-OBJECT is ready to fire   0.00
ID: 29 ORDER: 1   GI-DROP-OBJECT is ready to fire -10.00
ID: 2 ORDER: 2   gi-not-processed is ready to fire -100.00
**********************************************

GET-OBJECT had fired
FIRE  28          GET-OBJECT:  f-3  f-5  f-19  f-26 .
monkey grabs the bananas
<== Activation   28 GET-OBJECT:  f-3  f-5  f-19  f-26 .
<== Activation    2 gi-not-processed:  f-5 .
<== Fact f-5: (goal-is-to active holds bananas)
<== Activation   29 GI-DROP-OBJECT:  f-23  f-26 .
<== Fact f-26: (monkey t2-2 ladder blank)
==> Fact f-27: (goal-is-to satisfied holds bananas)
==> Fact f-28: (monkey t2-2 ladder bananas)
==> Activation   30 GI-DROP-OBJECT:  f-27  f-28 .
==> Activation   31 satisfy-hunger:  f-28 .


     FACTS IN THE DATABASE
```

```
**********************************
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 10; order: 3 (goal-is-to satisfied on floor)
id: 12; order: 4 (goal-is-to satisfied walk-to t9-5)
id: 14; order: 5 (goal-is-to satisfied holds ladder)
id: 17; order: 6 (goal-is-to satisfied walk-to t2-2)
id: 19; order: 7 (object ladder t2-2 light floor)
id: 20; order: 8 (goal-is-to satisfied move ladder t2-2)
id: 23; order: 9 (goal-is-to satisfied holds blank)
id: 25; order: 10 (goal-is-to satisfied on ladder)
id: 27; order: 11 (goal-is-to satisfied holds bananas)
id: 28; order: 12 (monkey t2-2 ladder bananas)
**********************************
**********AGENDA**********
ID: 31 ORDER: 0   satisfy-hunger is ready to fire   0.00
ID: 30 ORDER: 1   GI-DROP-OBJECT is ready to fire -10.00
**********************************************

satisfy-hunger had fired
FIRE   31      satisfy-hunger:  f-28 .
Monkey Eats the Bananas
<== Activation   31 satisfy-hunger:  f-28 .
<== Activation   30 GI-DROP-OBJECT:  f-27  f-28 .
<== Fact f-28: (monkey t2-2 ladder bananas)
==> Fact f-29: (monkey t2-2 ladder blank)
==> Activation   32 GI-DROP-OBJECT:  f-23  f-29 .


     FACTS IN THE DATABASE
**********************************
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 10; order: 3 (goal-is-to satisfied on floor)
id: 12; order: 4 (goal-is-to satisfied walk-to t9-5)
```

id: 14; order: 5 (goal-is-to satisfied holds ladder)
id: 17; order: 6 (goal-is-to satisfied walk-to t2-2)
id: 19; order: 7 (object ladder t2-2 light floor)
id: 20; order: 8 (goal-is-to satisfied move ladder t2-2)
id: 23; order: 9 (goal-is-to satisfied holds blank)
id: 25; order: 10 (goal-is-to satisfied on ladder)
id: 27; order: 11 (goal-is-to satisfied holds bananas)
id: 29; order: 12 (monkey t2-2 ladder blank)
****
**********AGENDA**********
ID: 32 ORDER: 0 GI-DROP-OBJECT is ready to fire -10.00
****

GI-DROP-OBJECT had fired
FIRE 32 GI-DROP-OBJECT: f-23 f-29 .
==> Fact f-30: (goal-is-to active holds blank)
==> Activation 33 gi-not-processed: f-30 .

FACTS IN THE DATABASE
****
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 10; order: 3 (goal-is-to satisfied on floor)
id: 12; order: 4 (goal-is-to satisfied walk-to t9-5)
id: 14; order: 5 (goal-is-to satisfied holds ladder)
id: 17; order: 6 (goal-is-to satisfied walk-to t2-2)
id: 19; order: 7 (object ladder t2-2 light floor)
id: 20; order: 8 (goal-is-to satisfied move ladder t2-2)
id: 23; order: 9 (goal-is-to satisfied holds blank)
id: 25; order: 10 (goal-is-to satisfied on ladder)
id: 27; order: 11 (goal-is-to satisfied holds bananas)
id: 29; order: 12 (monkey t2-2 ladder blank)
id: 30; order: 13 (goal-is-to active holds blank)
****
**********AGENDA**********

ID: 33 ORDER: 0 gi-not-processed is ready to fire -100.00
****

gi-not-processed had fired
FIRE 33 gi-not-processed: f-30 .
<== Activation 33 gi-not-processed: f-30 .
<== Fact f-30: (goal-is-to active holds blank)
==> Fact f-31: (goal-is-to not-processed holds blank)

FACTS IN THE DATABASE
****
id: 0; order: 0 (initial-fact)
id: 2; order: 1 (object couch t5-7 heavy floor)
id: 3; order: 2 (object bananas t2-2 light ceiling)
id: 10; order: 3 (goal-is-to satisfied on floor)
id: 12; order: 4 (goal-is-to satisfied walk-to t9-5)
id: 14; order: 5 (goal-is-to satisfied holds ladder)
id: 17; order: 6 (goal-is-to satisfied walk-to t2-2)
id: 19; order: 7 (object ladder t2-2 light floor)
id: 20; order: 8 (goal-is-to satisfied move ladder t2-2)
id: 23; order: 9 (goal-is-to satisfied holds blank)
id: 25; order: 10 (goal-is-to satisfied on ladder)
id: 27; order: 11 (goal-is-to satisfied holds bananas)
id: 29; order: 12 (monkey t2-2 ladder blank)
id: 31; order: 13 (goal-is-to not-processed holds blank)
****
Execution completed successfully