

3-1-1985

# Some Prototype Examples for Expert Systems v.1

K. S. Fu

*Purdue University*

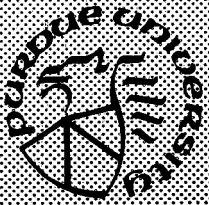
Follow this and additional works at: <https://docs.lib.purdue.edu/ecetr>

---

Fu, K. S., "Some Prototype Examples for Expert Systems v.1" (1985). *Department of Electrical and Computer Engineering Technical Reports*. Paper 534.

<https://docs.lib.purdue.edu/ecetr/534>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact [epubs@purdue.edu](mailto:epubs@purdue.edu) for additional information.



# Some Prototype Examples for Expert Systems

edited by  
K.S. Fu

Volume I

TR-EE 85-1  
March 1985

School of Electrical Engineering  
Purdue University  
West Lafayette, Indiana 47907

## Table of Contents

*Foreward*

### VOLUME 1

#### Part I -- Manufacturing

Chapter 1 Production Scheduling: A Sub-Aggregate Level Expert Scheduling Module <i>J. G. Maley</i> .....	1
Chapter 2 Expert System for Scheduling <i>D. Ben-Arieh</i> .....	19
Chapter 3 Expert Systems in Quality Control <i>Y. S. Chen</i> .....	106
Chapter 4 Deep Drawing Feasibility Expert System <i>G. Eshel</i> .....	167
Chapter 5 An Expert System For Machine Selection of FMS <i>S. Lan</i> .....	237
Chapter 6 PROLOG EXPERT: A Simple PROLOG based Expert System Framework for Synthesis, the BAGGER Problem as An Example <i>T. Sarjakoski</i> .....	270

### VOLUME 2

#### Part II -- Robotics

Chapter 7 MR1: An Expert System for Configuration of Modular Robots <i>D. Dutta and S. Joshi</i> .....	232
Chapter 8 RP -- An Expert System for Robot Programming <i>H. Zhang</i> .....	317
Chapter 9	

Spatial Planner, A Rule-Based System in Robotics  
*Y. L. Gu*.....354

**Part III -- Vision**

Chapter 10  
An Application of Expert System Approach in Detection of Boundaries  
Between Textures  
*K. B. Eom and C. Chatterjee*.....367

Chapter 11  
Stripe Pattern Interpreter and Stacker (SPIS): Expert System  
*H. S. Yang*.....477

**Part IV -- Management**

Chapter 12  
An Expert System For New Product Evaluation in Small Companies  
*Z. Xu and Q. Xue*.....496

Chapter 13  
Expert System for Inventory Models  
*K. Y. Tam and H. R. Rao*.....525

**VOLUME 3**

**Part V -- Structural Engineering**

Chapter 14  
Expert System for Damage Assessment of Existing Structures  
*X. J. Zhang*.....568

**Part VI -- Automated Programming**

Chapter 15  
An Experiment in Parallel Programming Environment:  
The Expert Systems Approach  
*K. Y. Wang*.....591

**Part VII -- Others**

**Chapter 16**

**A Prototype for an Expert System for Morphological Classification  
of Prehistoric American Pottery**

*C. Tsatsoulis and K. S. Fu*.....625

**Chapter 17**

**Expert System for Contract Bridge Bidding**

*L. Y. Chang and C. F. Yu* .....665

**Chapter 18**

**Air Flight Scheduler Expert System**

*A. J. Vayda and W. Y. Kim*.....698

**Chapter 19**

**Diet Expert System in Hospital**

*L. Chang and S. J. Lin* .....734

## FORWARD

This report consists of the nineteen term project reports for the graduate-level course EE695G "Expert Systems and Knowledge Engineering", which was offered for the fall semester of 1984 in the School of Electrical Engineering. The purpose of the term project is to provide each student an opportunity of designing and implementing a prototype expert system. The application area of each of these expert systems was selected by the student(s) working on the projects. This report is published for the purpose of documenting these results for future reference by the students of the above-mentioned course and, possibly, other workers in expert systems.

The nineteen reports are grouped into seven parts based on their application domains. Part I - Manufacturing consists of six reports, and Part II - Robotics contains three. Two reports in each of Part III - Vision and Part IV - Management, and one in each of Part V - Structural Engineering and Part VI - Automatic Programming. The last part, Part VII - Others, consists of four reports with different applications.

I would like to thank Mr. Edward K. Wong for his valuable help in putting the materials together for this report.

K. S. Fu  
Instructor, EE695G  
February 1985  
Lafayette, Indiana

\*\*\*\*\*

# PART I

## *Manufacturing*

---

**Chapter 1**

**Production Scheduling: A Sub-Aggregate Level Expert Scheduling Module**

*J. G. Maley*



## Production Scheduling:

### A Sub-Aggregate Level Expert Scheduling Module

---

James G. Maley

---

#### 1. Introduction

##### 1.1 Problem Statement

The problem to be solved within the context of EE6950 is the development of an expert system which schedules a production environment according to a user specified set of performance measures. The system under consideration will not focus on the real-time scheduling of parts through this production environment but rather take the more aggregate view of a shift work schedule. Encompassed by the scheduler in question are: the integrated performance measures, the system status changes on a shift level, the input parameters or required due dates, and capacity for large-scale implementation. This system will be a generic representation of a manufacturing system now in use by the AMP Corporation with specific test cases using their system.

##### 1.2 Problem Motivation

Within the production systems research area, the exact solution to the machine scheduling problem has eluded researchers for

a number of decades. Because of the difficulty of the problem, numerous heuristic methodologies have been implemented in actual production environments. These methods often require humans to solve parts of the problems. In past years this would suffice due to the fact that a feasible schedule was better than no schedule at all and a human scheduler could develop such a schedule. Also, the human scheduler would become an expert at determining the interrelationships which would provide 'good' schedules. Today, however, the competitive initiative of foreign manufacturers has required a new look at scheduling. If 'better' schedules are possible, then production can proceed at a greater level of efficiency which could result in a more competitive corporate production system. Thus the motivation for developing a production scheduler that more closely approaches the optimal solution than current methods.

### 1.3 Research Overview

The expert system scheduler developed as a part of the requirements for EE6950 will be incorporated into the larger framework described herein. To rephrase this, the work described in this paper is only a part of the following more general model. Figure 1 represents the present concepts for the configuration of the system. The three inputs; orders, local data, and events; are factors which change the status of the production facility and thus affect the operation of the scheduler. These inputs have the magnitude of frequency change specified. Note that the

'events' is a continuously changing or real-time varying input to the system. At the level of scheduling under consideration in this research, the 'events' data will only be incorporated into the schedule at the next shift. Thus a different or an extended system will be required to incorporate the data as it becomes available.

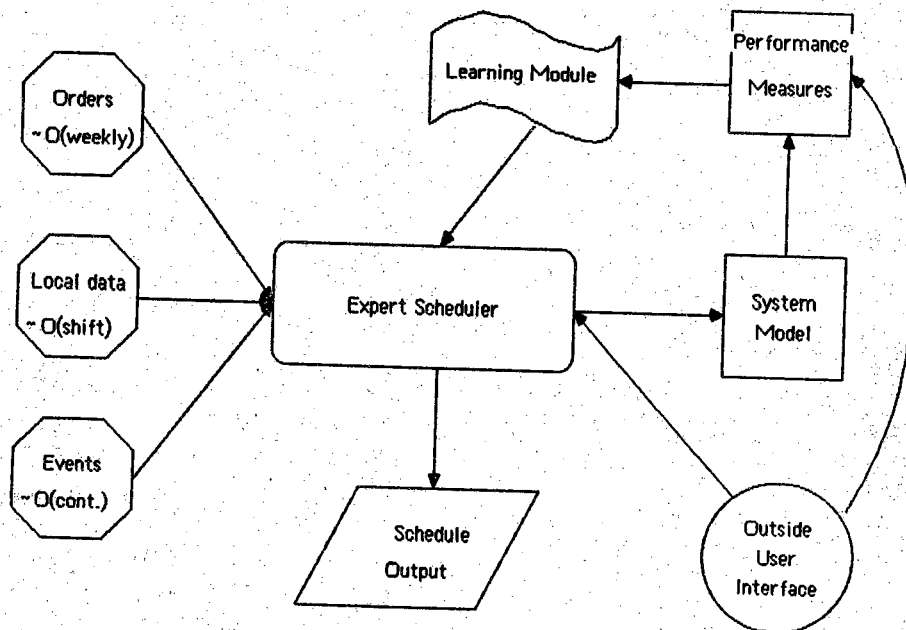


Figure 1. System Conceptualization

The expert scheduler, the focus of this research, is the core or the heart of the complete system. With the ability to accept various levels of input, the scheduler is a flexible piece of the overall system. The implemented model currently takes only information on weekly orders to schedule the machines involved in the AMP manufacturing facility. The primary reason for this

limited implementation is the lack of a resident expert on the entire system. Dr. James J. Solberg was the expert knowledge source for the current scheduling process. His expertise stems from his direct contact with the manufacturing facilities at AMP. He was very knowledgeable on the more aggregate scheduling process as it actually occurs at the various facilities.

Also included in the system shown in figure 1 are the system modeler - perhaps a simulation, the user modifiable performance measurement criteria, and some type of learning module. The aspect of this system of highest research interest is the learning module. At this early stage of work, very few ideas of its structure have been generated. Hopefully, the work involved in knowledge representation and reasoning will provide a basis for further work in the learning area. The schedule output is self-explanatory - it is just the resulting schedule from the system. The last major portion of the system is the user interface. This comes into play at both the performance criteria definitions and the schedule output. The former because the user may decide that different information is more important than during the previous week. The latter because the user may want to know why the system developed the schedule that it did.

## 2. System Organization

### 2.1 Broad Overview

Appendix 1 shows the flow diagram of the scheduling system implemented for EE695g. Because of the nature of scheduling, a large number of computations are required. As such, complete use of LISP, PROLOG, or a production system language (OPS5, YAPS, ...) would not be efficient. Therefore, an integrated system utilizing the control structure of UNIX was decided upon as the proper system. An initial entry of the needed data was entered into a standard data file. This file was processed by a production system implemented in OPS5 to modify the records according to standard rules obtained from the "expert". The resulting data was transformed using a UNIX sort utility into the actual scheduling routine used by AMP's experts. This routine is a modified if-then set of rules programmed in FORTRAN-77 because of FORTRAN-77's quicker computation time as compared to LISP. Finally, the after scheduling one week's worth of jobs, the resulting schedule is outputted to a data file. If at some point new entries are added to the system, then the loop would repeat itself as shown in the diagram.

## 2.2 Knowledge Base

As was referred to above, production systems were chosen as the knowledge representation scheme for this expert system. The IF-THEN rules of production systems have almost become standards of expert system implementations. Success of projects such as RI (or XCON), DENDRAL, PROSPECTOR, and PUFF have shown that production systems are a practical approach to take in developing a working expert system. Advantages in production system's such as modularity and uniformity also assisted in the determination of using this method of representing the expert knowledge captured in this work.

## 2.3 Inference Mechanism

With the two distinct part of the scheduling expert system

come two separate inference mechanisms. The initial production system is implemented in OPS5 developed at Carnegie-Mellon University and uses its inference scheme. This scheme is based upon a recency ordering of the productions. When a production is fired, it is tagged with a "time" which is used for conflict resolution. A complete description of the inference mechanism can be found in the "OPS5 User's Manual."

The section of the expert system programmed in FORTRAN-77 is basically a hierarchical structure with user defined parameters. The parameters permit the user to determine the depth of a search through the data to be applied to the rules. The rules themselves are applied in sequential order to the data set from the OPS5 section of the system.

### 3. Knowledge Acquisition

#### 3.1 Where the Knowledge Came From

The knowledge acquisition stage of the development of this scheduler expert system was performed during the semester with an "expert" about the AMP manufacturing system. Dr. James J. Solberg, Professor in Industrial Engineering, has visited AMP sites and is well versed upon the subject of scheduling in the AMP production cells. Because of the financial impossibility of traveling to an AMP location in person, Dr. Solberg volunteered to be the resident expert. Please remember that he is familiar with

both the system and the scheduling literature.

### 3.2 Observations and Conclusions

After numerous discussions with the expert, the following scenario was determined to be the usual process for scheduling production runs at the AMP facility:

When an order arrives at the manufacturing supervisor's office, a number of specific items are closely checked before the order is sent to the production scheduler. The cost of the order (directly related to the profit of the order), the size of the order, the company who is ordering, and whether or not the company is asking for a special rush job (or a favor) are each taken into consideration. After which, the order's due date is modified so as to change the priority of the job. An example being, if the order is worth more than \$10,000 then lessen the due date by 2 days. The reasoning behind this philosophy is that important jobs cannot afford to be late.

Once the due dates have been modified, scheduling takes place. The main thrust of the scheduling is to maximize the machine utilizations. Such a philosophy has developed due to the corporate policy of evaluating the various plants on their overall machine productivity. A second objective of the scheduling process is to minimize the lateness of jobs. In other words, try to get each job done on time. This is carried out by determining the slack in the system. Here slack refers to the due date minus the scheduled finish date. The production scheduler looks through an ordered list of the modified due dates and tries to pick the jobs which will result in the least change over time from the now scheduled on the system. This process permits the change overs to be minimized and thus let the machines run longer to raise their average utilization. The expertise involved in the scheduling process is in the determination in how far to search for the best job to schedule next.

### 3.3 Rules

The rules that were ascertained from the AMP scheduling expert are listed below:



IF the company is IBM or HP or DEC

THEN the order has preference and reduce the due date by 5 days

IF the company is CDC or Apple

THEN the order has slight preference and reduce the due date by  
3

IF the company is Honeywell

THEN the order has no preference and increase the due date by 3  
days

IF the company is not (IBM, HP, DEC, CDC, Apple, or Honeywell)

THEN the order is left as is

IF the company is granted "a special favor"

THEN the order has its due date reduced by 3 days

IF the order is worth less than \$10,000

THEN it's a small order, don't worry about it, increase the due  
date by 10 days

IF the order is worth between \$10,000 and \$100,000

THEN it's a good order and schedule as is

IF the order is worth more than \$100,000

THEN it's a priority order, reduce the due date by 3 days

IF the company is IBM, HP, CDC, or Honeywell and the number to produce is between 1000 and 5000 parts

THEN split the order into two equal sized parts, one with the current due date and one with the due date increased by 5 days

IF the company is IBM, HP, CDC, or Honeywell and the number to produce is over 5000 parts

THEN split the order into three parts, two half the size of the first and increase the smaller orders' due dates by 5 and 10 days respectively

#### 4. Experimental Results

##### 4.1 Capabilities

The expert scheduler created for EE695g is capable of han-

dling up to 100 jobs with little difficulty. In order to increase the system's ability beyond this point the array structure of the FORTRAN-77 section of the program must be modified. Currently the memory requirements of the FORTRAN-77 code are the limiting factors. A number of different examples have been run using the current configuration of the expert system; all with positive results. An initial goal that was set forth for the project was to develop a user friendly interface. Unfortunately, time constraints did not permit this stage of development to be undertaken.

#### 4.2 Demonstrative Examples

Shown in figure 3 is an example input record for the due-date modification section of the expert system (implemented in OPS5). The required information includes a job order number, the part number being ordered, the name of the company, the cost of the order, the size of the order, the current due date and whether or not the job is a special job (a favor) or not. By referring to the rules listed above, the set of data shown in the figure provide all the necessary information to modify the current due dates. Using these two records, shown in figure 3, along with eight others for a total of ten records, an example problem was formed and executed.

Figure 4 shows the output of the OPS5 due date modification routine after it has been sorted by the modified due date. This data is used as the input data stream for the FORTRAN-77 segment

702448	(order number)
66654	(part number)
IBM	(company name)
20000	(order cost)
1000	(number to produce)
20	(due date)
20	(due date)
no	(is this job a favor?)
702449	(order number)
66245	(part number)
CDC	(company name)
30000	(order cost)
5000	(number to produce)
20	(due date)
20	(due date)
yes	(is this job a favor?)

Figure 3. DPS5 Input Data Example

of the expert system. Note in figure 4 that the order numbers have been changed. This change takes place when a large order is split into smaller orders.... a new order number is created. The key aspect of this figure is the modified due date ordering of the records.

Figure 5 is the final output file for the system. It shows not only the orders and part numbers, but the machines that the parts were scheduled upon, the start and finish times of the parts as well as the resulting set-up time by scheduling the parts on the machines. Note that the order of the jobs in figure 5 does not correspond directly with the order of the jobs in figure 4. This shows that the system did modify the order that the jobs were processed so that the machines were subject to as little set-up changes as possible.

Order #	Part #	Quantity	Mod Due Date	Due Date	Cost	Company
702449a	66245	2500	14	20	30000	CDC
702448a	66654	500	15	20	20000	IBM
702452	66624	2000	15	20	25000	DEC
702451a	66624	3500	17	25	25000	HP
702449b	66245	1250	19	25	30000	CDC
702453a	208022	500	20	20	2000	IBM
702448b	66654	500	20	25	20000	IBM
702450a	207076	3500	20	25	25000	HP
702457a	207076	2000	20	25	80000	HP
702451b	66624	1750	22	30	25000	HP
702449c	66245	1250	24	30	30000	CDC
702453b	208022	500	25	25	2000	IBM
702456a	66244	1500	25	28	75000	CDC
702450b	207076	1750	25	30	25000	HP
702454a	66077	1250	25	30	22000	IBM
702457b	207076	2000	25	30	80000	HP
702451c	66624	1750	27	35	25000	HP
702455a	66624	1250	30	30	35000	HONEYWELL
702456b	66244	1500	30	33	75000	CDC
702450c	207076	1750	30	35	25000	HP
702454b	66077	1250	30	35	22000	IBM
702455b	66624	1250	35	35	35000	HONEYWELL

Figure 4. Sorted Input for FORTRAN-77

#### 4.3 Performance Evaluation

The computational aspects of the scheduling expert system are not addressed in this report because of insufficient research time to perform such a task. Evaluation of the expert knowledge of the system, however, is now discussed. In order to evaluate the performance of an expert system, one would follow the same procedure that one would use to evaluate a human expert. The tests possible are empirical and statistical tests. The empirical tests involve using a set of examples and observing how well the system performs on these examples. The statistical tests require a large number of examples to be executed by the expert

Order #	Part #	Quantity	Mod Due Date		Start	End	Mach	Set-up
			^	^				
702449a	66245	2500	336	480	0	34	1	9
702448a	66654	500	360	480	0	14	2	9
702452	66624	2000	360	480	0	29	3	9
702451a	66624	3500	408	600	0	44	4	9
702449b	66245	1250	456	600	0	21	5	9
702453a	208022	500	480	480	0	19	6	9
702448b	66654	500	480	600	14	19	2	0
702450a	207076	3500	480	600	19	89	2	0
702453b	208022	500	600	600	19	31	6	0
702449c	66245	1250	576	720	21	43	5	0
702451c	66624	1750	648	840	29	68	3	0
702457a	207076	2000	480	600	31	110	6	11
702451b	66624	1750	528	720	34	89	1	9
702456a	66244	1500	600	672	43	73	5	0
702454a	66077	1250	600	720	44	90	4	9
702450b	207076	1750	600	720	68	182	3	11
702457b	207076	2000	600	720	73	228	5	16
702455a	66624	1250	720	720	89	151	1	0
702450c	207076	1750	720	840	89	139	2	0
702456b	66244	1500	720	792	90	136	4	16
702454b	66077	1250	720	840	110	156	6	9
702455b	66624	1250	840	840	136	157	4	9

Figure 5. System Output

system. Then, using the data gathered, the system is statistically evaluated.

Due to lack of real problems at this point in time, no extensive testing of the scheduling expert system was performed. A small empirical testing of the system did show that on the limited problems, the expert system performed as expected by creating product schedules that tried to maximize the machine utilization subject meeting due dates.

## 5. Conclusion and Discussion

The expert scheduler developed for EE695g is definitely a prototype for an actual system that could be implemented in the actual production environment at the AMP Corporation. The system must be enlarged to include more rules and larger working memory to handle the problems used in real production settings. With this enlargement of capacity, the expert system will be able to compete with the current human experts now scheduling the production system. From the results of this prototype, an expert system has been shown to have the ability to handle the expertise of human schedulers. This ability in itself is reason enough to press on with the expansion of the current expert system to upgrade it to the implementation level.

6. List of References

Baker, Kenneth R., Introduction to Sequencing and Scheduling, New York: John Wiley and Sons, 1974.

Conway, Richard W., William L. Maxwell, and Louis W. Miller, Theory of Scheduling, Reading, Mass: Addison-Wesley, 1967.

Forgy, Charles L., "OPS5 User's Manual," Department of Computer Science, Carnegie-Mellon University, 1981.

Hayes-Roth, Frederick, Donald A. Waterman, and Douglas B. Lenat eds., Building Expert Systems Reading, Mass: Addison-Wesley, 1983.

Solberg, James J., Personal Interview, Purdue University, Fall 1984.

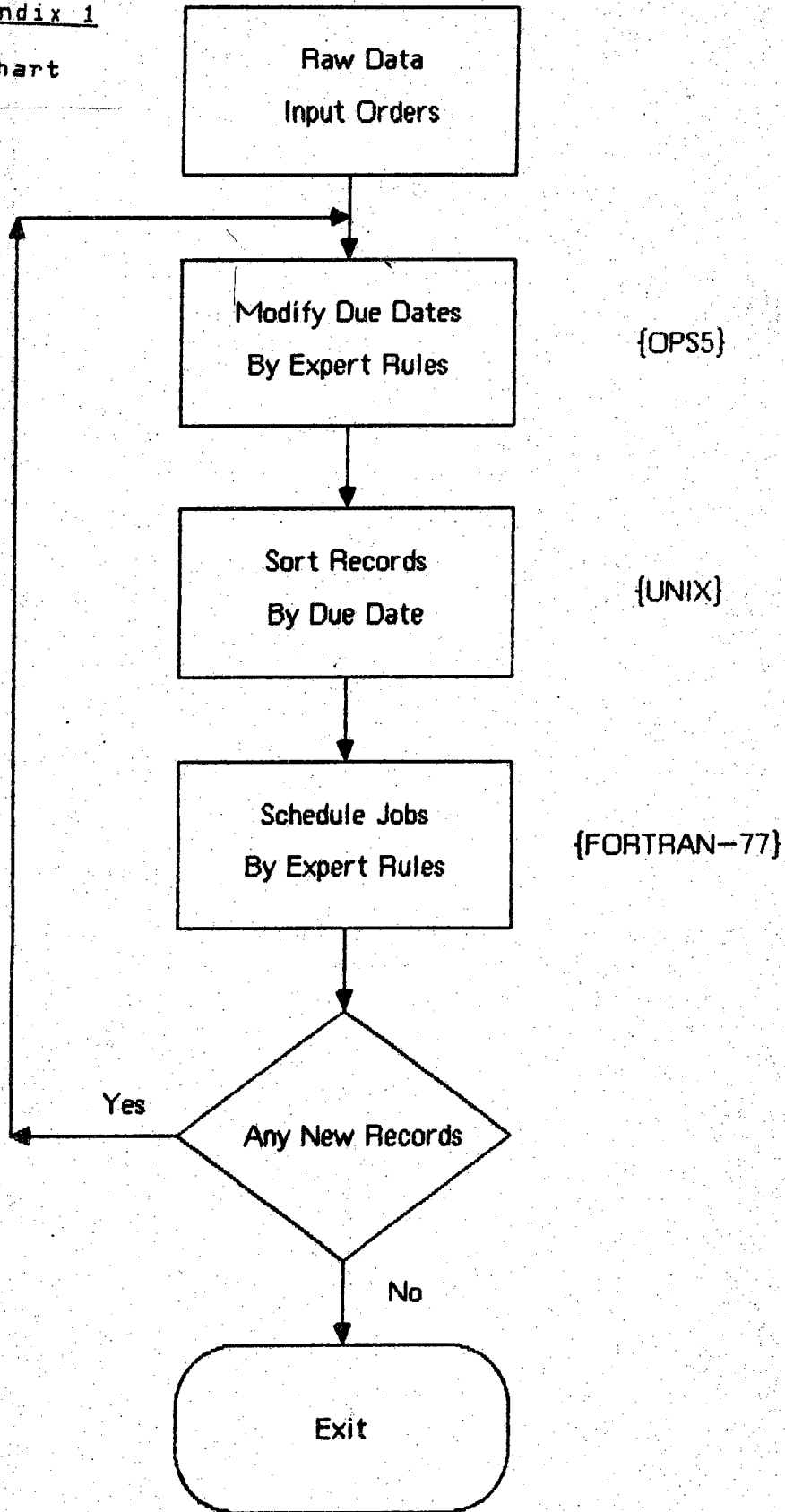
"Stamping Performance Data Output," AMP Corporation Computer Scheduling Data, 1984.

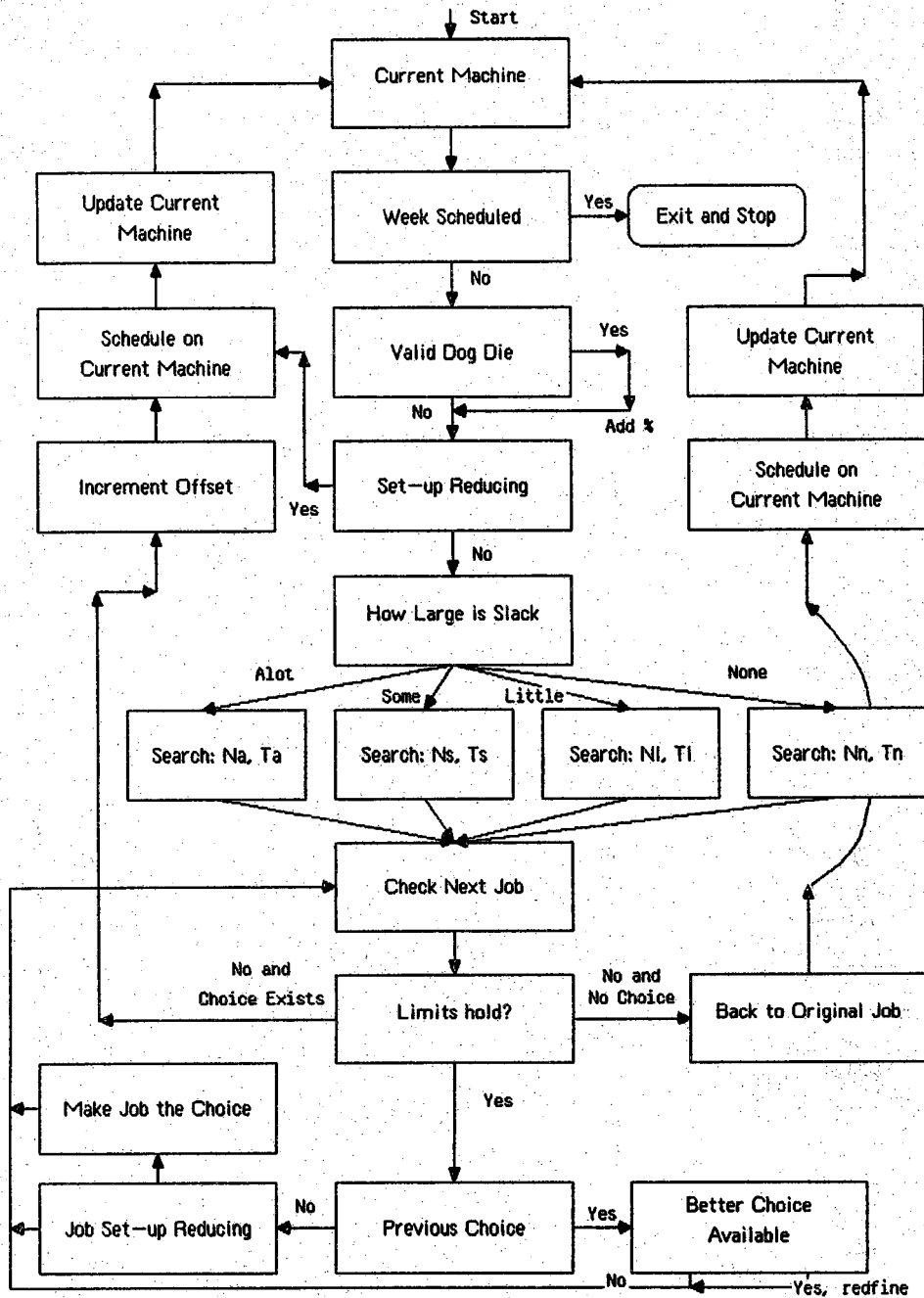
Wilensky, Robert, LISPcraft, New York: W.W. Norton and Company, 1984.

Whinston, Patrick Henry, Artificial Intelligence, Reading, Mass: Addison-Wesley, 1984.



7. Appendix 1  
Flow Chart





---

Chapter 2

Expert System for Scheduling

*D. Ben-Arieh*

EXPERT SYSTEM FOR SCHEDULING

David Ben - Arieh

1. THE EXPERT SYSTEM AND THE PROBLEM DOMAIN

The expert system in this project has to control a production facility that feeds an assembly station. The complete system consists of autonomous cells (CMS), and an assembly station, and the expert system task is to supervise this complex system.

1.1 A DESCRIPTION OF THE SYSTEM

The production system consists of computerized manufacturing cells, that can perform a large variety of processes with minimal set-up time. Parts are introduced into the system randomly or by demand and after being processed the parts are fed to an automated assembly station. Parts can choose almost any machine to perform the various processes in order to reach the assembly station on time. The routing problem in the production area is therefore a problem of dynamic routing, in a job-shop environment with multi-purpose machines. This problem is a combinatorial problem which does not have a closed analytical solution.

Because of the difficulty in controlling the system with "conventional" methods, an expert system is suggested in this project. The expert system will use knowledge about the current state of the shop, capability of every machine and the end product structure (assembly tree and processing times), in order to decide upon the best behavior of the system (route the parts, assemble the product, etc).

The control problem of the production facility has a similar nature to the general decision process. This process involves the following steps which will be performed by the expert system:

1. Identifying the problem.
2. Establishing feasible alternative actions.
3. Evaluating the outcomes of the alternative actions.
4. Selecting the "best" action.
5. Implementing the chosen alternative.

## 1.2 WHY EXPERT SYSEM

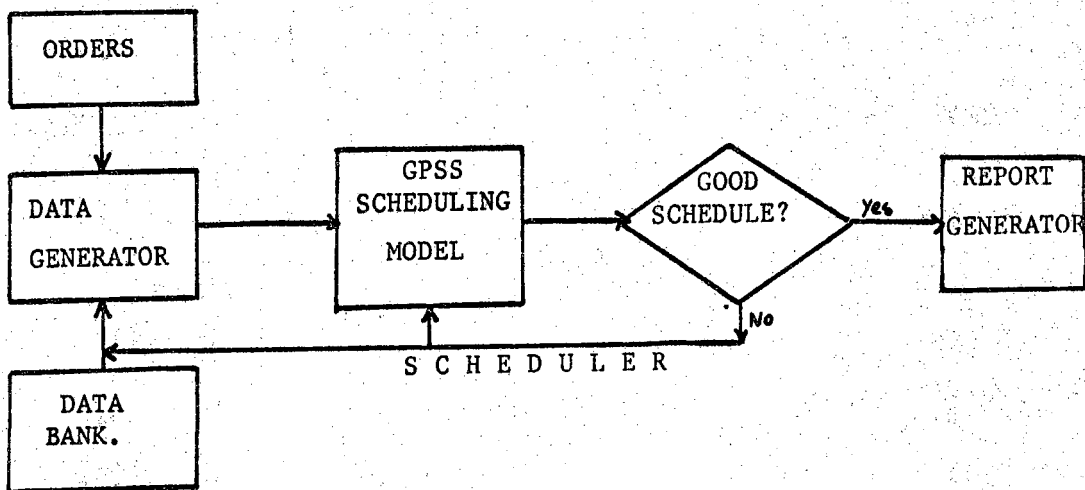
The problem of scheduling a job shop is a very complex one, as will be discussed later. Algorithms for solving this problem analytically using a computer do not exist, or consume too much time to be practical.

The next step in order to give a good solution was an interactive scheduling system that combines the computer computation power, with the reasoning of a human scheduler. Not much research has been done on such symbiotic systems, but still some results can be shown [Godin 1978].

It has been found that an interactive scheduler can get a better performance of the system than an off line scheduler with a fixed policy, even with a simple not sophisticated policy. Another result showed that a scheduler with some "look ahead" capability perform better than a scheduler without predictive tools. Suresh in his research [Suresh 1974], built a system that used simulation to help the scheduler in predicting the effect of his decision on the system.

His system has the following structure:

Figure 1. The structure of an interactive scheduler



Other similar experiments can be seen in Ferguson (1969) and Conner (1972). Although many of the experiments were not interested in the scheduling performance, but more in learning about the decision maker, the results gathered can be combined into the conclusion that interactive intelligent scheduler with predictive tools, is superior to any practical solution available.

The next step then will naturally lead to a computerized intelligent scheduler that has the knowledge and understanding of qualitative measures of the schedule as the human has, with the predictive and computational capability of the computer. A part of this idea is implemented in this project

It seems adequate to end this part with Simon and Newell [Simon, Newell 1958], enthusiasm (overenthusiasm?) about a new era that begins in which computers will deal with judgemental and intuitive tasks.

## 2. SCHEDULING - REVIEW

Scheduling is defined by Baker [Baker 1974] as "the allocation of resources over time to perform a collection of tasks". From such a general definition it is clear that the scheduling domain contains a wide variety of problems.

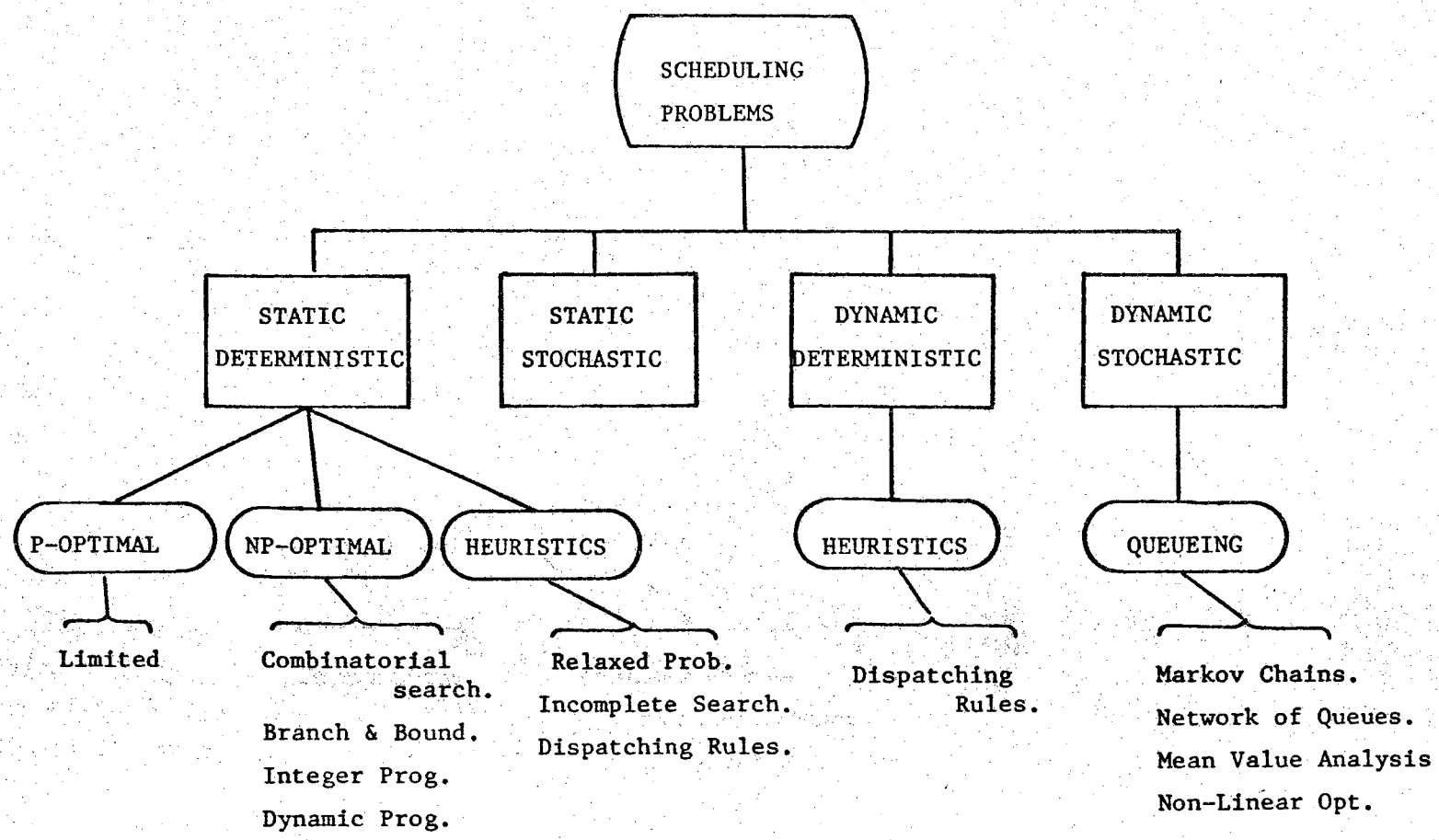
In order to narrow down the domain a problem classification is required, and the first approach is based upon the data variability, and data time dependency [King and Spachis 1980].

- i. Data variability: The problem is deterministic if all of the data involved is deterministic. The scheduling problem is stochastic if any of the data is stochastic.
- ii. Data time dependency: The problem is static if none of the initial data changes over time, otherwise the problem is dynamic.

This classification is depicted in figure 2.



Figure 2. Classification of scheduling problems and solution methods

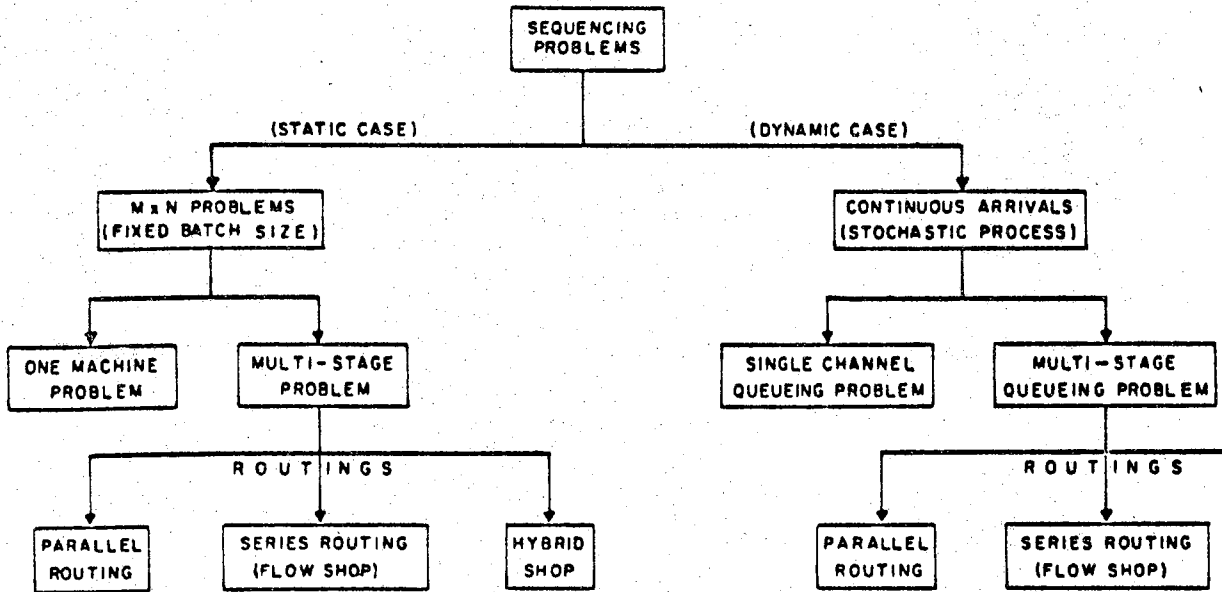


Another classification of scheduling problems is according to three factors [Day and Hottenstein 1970]:

- i. Number of components comprising a job (single component jobs, or multi-component jobs).
- ii. Production resources possessed by the shop (machines, labor and machines).
- iii. Jobs arrival for processing (all jobs are available initially, or jobs arrive continuously).

The last factor define the problem to be static or dynamic problem. The number of components factor determine the the nature of the job route. The production resources make the problem multi stage, or one machine problem. This classification is depicted in figure 3.

Figure 3. classification of scheduling problems



## 2.1 SOLUTION METHODS

### 2.1.1 STATIC DETERMINISTIC PROBLEMS

An optimal solutions with an efficient polynomial algorithms exist only for a limited set of static problems:

1. Single machine scheduling with a finite number of jobs.
2. Two machines problems with flow shop structure or one operation on each machine.
3. M-machines problems with severe limitations (two jobs, or identical machines with a unit process time, etc).

For more complicated static problems the approaches commonly used are:

1. Combinatorial approach. This approach is based on the changing of one permutation to another.
2. Mathematical programming. This includes linear programming, dynamic, convex, and quadratic programming, integer programming, branch and bound, networks of flow and the like.
3. Heuristic approach (approximate solutions). This approach when carried to completion guarantee an acceptable solution if one exists, or the knowledge

that none exists.

Some of the methods used are:

- i. Exact solutions to relaxed problems.
- ii. Incomplete search.
- iii. Ad hoc decision rules (dispatching rules).

### 2.1.2 DYNAMIC SCHEDULING

In the deterministic case, usually scheduling a system (especially a job-shop system) is done with dispatching rules that decide in real time which job to choose from a queue. However this approach do not consider multiple routes for parts, or assembly precedence relations.

In stochastic problems the common approach is by using queueing theory and networks of queues or operational analysis of queueing models [Denning and Buzen 1978]. The typical assumptions for this class of problems are:

1. The system can be modeled by a stationary stochastic process.
2. Jobs are statistically independent.
3. Jobs steps from device to device follow a Markov chain.

4. The system is in stochastic equilibrium.
5. Exponential service times.
6. First come first serve queue discipline.

Because of the stochastic nature of the parameters in dynamic scheduling, Monte-Carlo simulation has been the principle tool of analysis. In this case some of the scheduling solutions use dispatching rules.

In general the solutions to this class of problems are impeded severely by the assumptions that prohibit dependence between the system state and the policy in use, there is no blocking allowed and no precedence constraints that a solution can consider.

### 3. CMS CONTROL PROBLEM

The control of a CMS can be analyzed from various points of view. Most of the control mechanisms decompose the CMS into hierarchical levels as mentioned in [Buzacott 1976]. In this paper the system is composed of three main levels:

1. Prerelease planning, deciding which jobs are to be manufactured by the system.
2. Input control, determining the sequence and timing of the release of jobs to the system.
3. Operational control, controlling the movements of parts between the machines and other process-time decisions.

At each level of control the physical configuration and the decisions made at a higher levels set constraints on the alternative actions. It is also stated that in each level it is possible to generate a better solution if the 'rule' is more informed. It can also be shown that using information in the input control level gives a better efficiency of the system than if rules are only used in the prerelease level.

It is important to notice that the resource limitations causes the basic need for information to consider solution. If for example there is no space limitation

within the CMS then the decision on release of jobs can be made without using any information from the system at all. Since usually the system is limited in space and every machine has a small buffer, a better solution can be generated using the detailed level of information - the operational control level.

A point of view different from the hierarchical control strategy can be seen in the the paper of Kusiak (1984). In this paper the control of a CMS is performed by a management system, and the operational control is mainly interested in part scheduling. In this article the various levels in the hierarchy differ in the time horizon of the plans, and the lower level is the part scheduling. The solution to this problem determines the solutions to the tools, AGVs, pallets and the other resources scheduling.

J. Kimemia and S. Gershwin [24] developed a different structure of the control system of a CMS. In their work the tasks that are performed at each level are not mentioned, but the lowest level deals with dispatching the parts. In this work the hierarchical structure is implicitly assumed but has no influence upon the scheduling algorithm.

A different approach is found in [McLean et.al]. In this research the hierarchical control of the CMS con-



tains the following levels:

1. Facility control: The highest level which contains the design process and the management system (inventory, accounting etc ).
2. Shop control: This level is responsible for the real time management of resources distribution, and jobs schedules in the shop.
3. Cell control: The sequencing of batches of jobs through the workstations, and supervision of support services such as calibration and material handling.
4. Workstation control: coordination between the activities in a workstation floor equipment (robot, NC machine, storage buffer etc).
5. Equipment control: controls a particular piece of equipment on the shop floor.

### 3.1 SCHEDULING OF A CMS

In an attempt to find an analytic solution to the CMS routing problem, there is a need to examine the most similar family of scheduling problems : the job shop scheduling [Bellman].

In the job-shop scheduling problem there are  $n$  jobs, and  $m$  machines. Any job can be processed on each

of the  $m$  machines only once, and the order of the machines required to process each job  $i$ , is represented by a  $n \times m$  matrix with  $T_i$  as its  $i$ th row where

$$T_i = (i_{q1}, i_{q2}, \dots, i_{qm}).$$

The time to process each job on each machine  $M_q$ , defines an  $n \times m$  matrix with  $P_i$  as its  $i$ th row, where  $P_{i,q}$  is the processing time of job  $i$  on machine  $M_q$ .

The sequencing problem is a static problem that decides the order of all jobs on each machine in order to optimize an objective function, given the ordering matrix and processing time matrix for  $n$  jobs and  $m$  machines. The order of all jobs on machine  $M_q$  is expressed by  $S_q$ .

The assumptions that are taken in static job shop scheduling problems are:

1. All  $n$  job sets are available in the beginning.
2. No processing of any operation can be done by more than one machine.
3. Any operation starting to be processed cannot be interrupted.
4. There are no priority orders within jobs and each job has the same importance.
5. There is no limit on in-process inventory. Every

job can wait until the former operation is done.

6. Each job can be processed by each machine only once.
7. All  $m$  machines are available. Breakdowns or repair of any machine does not occur during the planning period.
8. The machines are independent of each other.
9. Processing time of each operation are given and are constant regardless of the order of processing.

It is clear that assumptions 1,4,5,6,7,8 are not valid in this case and violate the combined assembly and production model. This conclusions leads towards different solution methods, than the methods used in static job shop scheduling problems.

### 3.2 OTHER CONTROL METHODS FOR CMS.

In addition to the classical scheduling techniques a large class of control algorithms uses close networks of queues. This method uses theory developed by Jackson (1963) and Gordon and Newell (1967) models the system as a network of queues and parts that come out of one queue enter another one. The first direct application of queueing theory to FMS's is due to Solberg (1978). In this model there are the assumptions of: equilibrium

behavior, exponential service times, and infinite queue space in the system. This assumptions makes the model to have good agreement with those performance measures obtained from similar systems. Some of the other works that uses this theory are Stecke's doctoral work [Stecke 1981] that dealt with FMS detailed parts scheduling and a research by Buzen [Buzen, 1973] that showed ways of solving networks of queues.

Another model for FMS was suggested by Buzacott [Buzacott 1980], in which probabilities  $P_{ij,r}$  create transition matrix to route a part from class  $r$  from machine  $i$  to  $j$ . In this work process times and interarrival times are exponentially distributed. Some of the conclusions of the work are:

1. In FMS in order to have the best performance, jobs should have diverse routes available.
2. For jobs with some flexibility in the sequence of operations it is better not to fix this sequence at the pre-production planning level.
3. Common storage is superior to local storage because of control needs. Local storage should be used only if there is close control over the release of jobs to prevent blocking.

A similar work by Suardo [1979] uses queues network. This work assumes FIFO discipline in the queues,

exponential service times and infinite buffer size in the system (no blocking). This work does not give a closed form solution but computationally it is solvable. The model is stated as a non linear programming with linear constraints and convex objective. The problem converges to linear programming when the system approaches saturation.

Another approach to FMS scheduling is developed by Hilderbrant (1980) and is called mean value analysis. In this work the writer assumes FIFO discipline in the queues and steady state behavior. The model tries to minimize completion time of the production target under constraints of machines failures.

#### 4. AI APPROACH TO THE CONTROL PROBLEM

Some researchers distinguish four phases in the development of computerized manufacturing systems [Hatvany 1983]:

1. The first phase was that of direct computer control of groups of machine tools.
2. The second phase was that of flexible manufacturing systems equipped with automatic workpiece transport and changing devices, tool changers.
3. The third step has been defined as that of computer integrated manufacturing and consists of systems that integrate the design, process planning and production control to some extent.
4. The fourth and final phase is that of intelligent manufacturing systems that have the capability to solve problems without explicit algorithm available.

This approach emphasize the crucial role of AI in the manufacturing environment because the unstructured nature of the problems in this area.

Another work [Bullers 1980] looked at the control needs in the manufacturing environment. In this area there are three main levels of activities: the

strategic level, the tactical level and the operational level. It seems that the most demanding level is the operational level where problems are introduced dynamically and need to be solved as fast as possible. In this work the main task of an AI system is problem solving both in static and dynamic time domains. In the static time domain some of the necessary steps are: primitive problem procedure invocation which are procedure calls to the database that solve the problem. Some of the problems that this approach can solve are for example:

Are there any parts of type B in the system?

Another mode of the system is procedure invocation of a unique axiomatic procedure. In this case a unique procedure is invoked to reduce the goal problem into a set of primitive problems. An example for such a problem is to find the first operation for part A for example. A more difficult step to take is 'procedure invocation of a non-unique axiomatic procedure'. In this case the system treats problems that require selection of one of many procedures with the same name to reduce the goal problem to a set of primitive problems. An example to such a problem is: What operation is to be done next on part A?

The hardest problems to be solved involve 'procedure invocation of multiple, possibly non unique axiomatic procedures'. A problem of this type is: On

what machine should part A be scheduled for the next operation if the part needs a machine with the shortest processing time.

Solving the problems in a dynamic time domain is more difficult since the status of the system is changed with time.



#### 4.1 EXPERT SYSTEM: REVIEW

Expert system is a tool that belong to the artificial-intelligence field, and its objective is to solve problems that are difficult or impossible to solve numerically. Expert systems are problem solving computer programs that can reach a level of performance comparable to that of a human expert in some specialized problem domain.

Expert systems differ from regular application computer programs in the internal structure of the program: Application programs are basically composed of two elements:

- I. Specialize problem solving knowledge.
- II. Specific data of the problem.

On the other hand expert systems are composed of three parts:

- I. Knowledge base. A part that contains the general knowledge of the problem in the application domain.
- II. Global database. This part contains the facts known or inferred about a specific case. This is the working database.
- III. The control mechanism (inference procedure). This part contains the set of functions that control the

interaction with the users, and update the current state of the knowledge about the case in hand. The control system also decides which rule to apply next in order to solve the problem, and how to search for a solution.

#### 4.1.1 REPRESENTATION OF KNOWLEDGE

This topic is a crucial element in the expert system implementation. The representation of the knowledge is a commitment to a vocabulary, data structures and programs that allow the knowledge to be acquired and used.

There are three basic requirements on representation of knowledge in expert systems [Buchanan, Duda]:

- I. **Extendability:** The data structures and the programs must be flexible enough to allow extension to the knowledge base without forcing substantial revision.
- II. **Simplicity:** The data structures must be conceptually simple and uniform to achieve flexibility and ease of use, analysis display etc.
- III. **Explicitness:** Represent the items of knowledge explicitly to get easy debugging, inspection and understanding of the knowledge available at each

step of the solution.

In order to achieve the above goals three types of representation framework are used in expert systems:

- I. Rule base system (production system) [Davis and King].
- II. Frame based system (frames, semantic networks, scripts).
- III. Logic based system (first order predicate logic).

#### 4.1.2 THE CONTROL PROCEDURES

In the control level there are several methods that are used in expert systems [Nau 1983]:

- I. Propagation of constraints. In this problem-solving technique, the set of possible solutions becomes further and further constrained by rules or operators.
- II. Data drives control (forward system). In this case rules are applied whenever their left hand side condition is satisfied.
- III. Goal driven control (backward system). In this way of control, only rules that are applicable to some particular goal are applied. In this way the solu-

tion starts with the known goal and try to reach the initial conditions that are available in the system.

IV. Mixed strategies. In this way the system looks for a path that connects the goal with the initial state, by progressing from both ends of the problem.

V. Problem reduction. This technique converts the problem to an AND/OR tree, and the system then needs an AND/OR search for a solution. Several reduction steps can take place recursively in order to simplify the problem further.

VI. Generate and test. In this case the system generates states of the search space, and tests each in turn until it finds one satisfying the goal condition.

#### 4.1.3 THE EXPERT SYSTEM FOR SHOP FLOOR CONTROL

In the scheduling domain, the expert system tries to imitate the shop floor experienced worker, that knows from his experience how the machines and jobs interact, and route the incoming parts upon his judgement. A good experienced worker can find it difficult to explain, justify and quantify his decisions. This fact makes it very difficult to summarize this experience into an

expert system.

An interesting example for an expert system that schedules a job-shop environment is ISIS [Fox 1982, 1983]. This system uses an heuristic search approach in a constraint environment to achieve its objective. This system which is written in SRL (a knowledge representation language), schedules orders that arrive to Westinghouse Electric Corporation Turbine Component Plant. The objective is to meet due dates while satisfying the constraints in the plant. The constraints are divided into three main groups: In group one there are "organizational goals" like in process inventory, resources level, production level and shop stability (in global terms). The second group contains physical constraints (ability of machines etc), and in group three this system has the precedence relations and resource requirements. The scheduling decisions are made on the basis of current and future costs (lose of a customer if a job is late), and profits. The scheduling then is a constraint directed search, taking into account conflicts, importance of constraints and interaction between constraints. The system has a multi layer structure.

Another work that considers expert system for production control is found in Nof's work [Nof 1983].

#### 4.1.4 Classification of the expert system

Expert systems can be classified according to the tasks they perform [Stefik 1982], or the level of complexity that the problem has. This expert system main task is planning, but it also supply monitoring capability to the system (tool life, machine failure..). The key problems that systems with these tasks have are:

- I. Problems can be very large and complicated, and the consequences of actions are not well understood.
- II. Many details to take care of.
- III. Interaction between plans for different subgoals. This is one reason for the complexity of the system.
- IV. If the plan is to be carried out by multiple actors, coordination can become difficult. In the routing case coordination is required between machines, material handling devices, load-unload stations, bar-code readers, etc.
- V. The monitoring must be credible and the system must avoid false alarms.

The specific problem that this expert system must cope with has the following properties:

- I. A 'real time' solution is required to the routing problem, independent from the status of the system (how complicated or 'bad' things are..).
- II. Time varying data.
- III. Large solution space. This property is reduced by the problem reduction technique and the special search method used.
- IV. Evaluation for partial solutions. It is desired that the system will have evaluation function for partial solutions in order to prune undesirable solutions. In this case the evaluation function is a heuristic function that evaluate the solution by the waiting time of the assembly station, and the in process inventory in the system.

$$f(n,t) = 100 W(t) + EQ_i(t)$$

$W(t)$  = The expected idle time of the assembly system, caused by the shortage of part n.

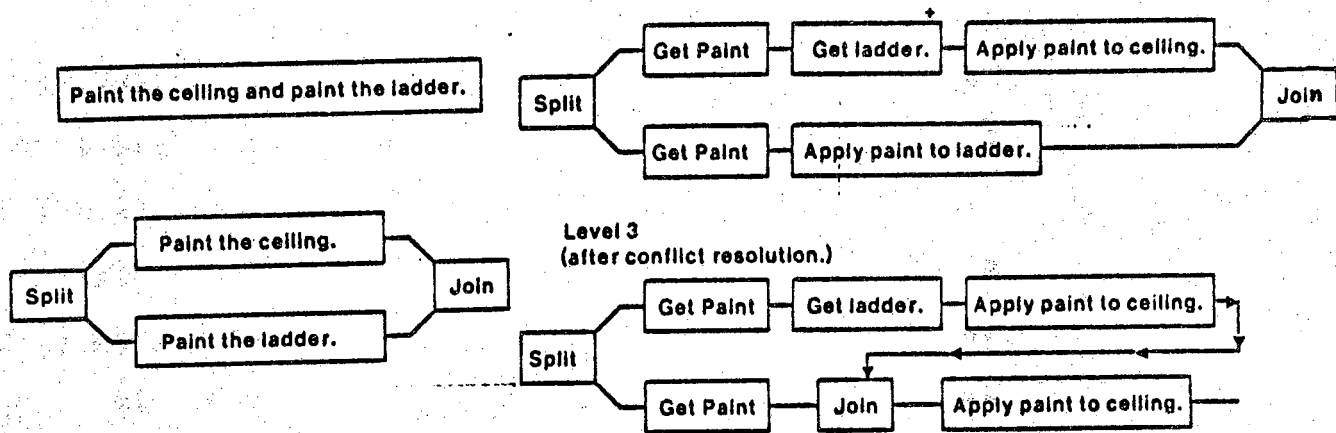
$Q_i(t)$  = The buffer size at time t, in machine i.

Since this function is heuristic, we may find better functions that have a better 'resolution' or separation between good and bad partial solutions.

- V. Interacting subproblems. This is the most severe problem in achieving an optimal solution to the

routing problem. An example to interacting subproblems can be found in the NOAH system [Stefik 1982]. The example can be described by the following figure:

Figure 4. example for interacting subproblems



In the context of parts routing, the system routes every part separately, without being aware of later routes that can interfere with the current one. For example the route of part A does not consider the route of part B (because it will take place after that of part A), but part B will block one of the machines that part A needs.

#### 4.2 PROLOG AND EXPERT SYSTEMS

PROLOG is a computer language whose name stands for PROgramming in LOGic. This language was initiated in



France at the university of Marseilles, by Alain Colmerauer and others. This version of PROLOG was improved by David Warren in the University of Edinburgh, who created a PROLOG interpreter for the DEC-10 computer [Hayes and Michie 1983].

If we consider steps in the development of "AI" languages the first such a work was implemented by Hewitt in PLANNER in 1971, and later on this language was the basic model for such works. Simpler versions were implemented by Sussman, Winograd and Charniak in MicroPLANNER, by Rulifson in GA4 and later on in Conniver (Sussman in 1972). However by 1972 these languages became known for being inefficient and hard to control, and so by 1975 the idea of such languages died in United States. The only language that was used was LISP that was developed in the fiftys as a general language. At that time PROLOG was created in Europe, and it seemed remarkably like PLANNER, however this language has attracted the user community and seem to be successful.

In a work by McDermott he describe the advantages and shortcoming of PROLOG [McDermott 1980] :

#### Advantages of PROLOG

1. It has a powerful pattern matching mechanism (better than in PLANNER ,GA4 or Conniver).

2. Data structures are very easily created according to the knowledge of interest. The pattern matching works the same on all kinds of terms.
3. The language is very efficient, not as compact as LISP but just as fast.
4. PROLOG supplies certain AI oriented features such as pattern matching and an assertional data base.
5. PROLOG is easier to learn and implement than LISP.

#### Disadvantages of PROLOG

1. The notion that PROLOG uses programming in logic is not true.
2. The unification process used in PROLOG is in most implementations not a real full unification (logically it is even incorrect).
3. It is claimed that PROLOG do not use "side effects" which is not really true.

In another research effort by Warren and Pereira in 1977, the writers compared PROLOG with LISP. The findings were that PROLOG uses a simpler syntax than LISP (it is more forgiving to syntax "errors"), and it ran about 1.1 to 2.6 times faster than LISP. From space point of view PROLOG was at least two times better than LISP. Other properties of PROLOG are better readabil-

ity, size of code and complexity (degree of nesting).

A more detailed survey compared PROLOG with INTER-LISP and FORTRAN [Mizoguchi 1983]. This research compared four expert systems written in the three languages. The expert systems were APLICOT (PROLOG), EXPERT (FORTRAN), EMYCIN (Inter-Lisp) and ADIPS (Inter-LISP). All the systems were used for a mission of fault diagnosis in a reactor cooling system. The results show that the PROLOG code was 5 times shorter than the INTER-LISP and 10 times shorter than the FORTRAN, while the response time was similar in all three systems.

Another example for a successful expert system written in PROLOG is a system developed in Lockheed for tactical data fusion [Rauch et. al 1982]. This system gets as input high data rates from sophisticated military sensor system and outputs decisions about the military situation.

#### 4.3 Knowledge Based Simulation

The concept of using the knowledge embedded in a knowledge based system in order to achieve a very detailed simulation is new and not much has been done in the area. One implementation is ROSS [Klahr and Faught 1980] which is used to create a very large scale simulation of a military air battles. This work was written in a language called Director which has properties like

pattern matching and IF-THEN rule structure (very similar to PROLOG). This system has about 75 behavioral rules, 10 types of entities and it has been run with up to 250 individual entities.

Another example that uses rule base system was written on a PROLOG basis (T-PROLOG). This system was developed in order to demonstrate the combination of simulation with operative problem solving available through the backtracking mechanism of PROLOG [Futo and Szeredi 1984]. This system was used to run simple simulations that required some conditions to become true at the end of the simulation. Since during the simulation there was no information about the conditions the simulation used to advance blindly and then backtrack if the conditions were false.

## 5. THE IMPLEMENTATION

It is clear that scheduling a production with assembly process is a very complex problem, especially if a real time solution is expected. It is desired that the algorithm should be centralized in order to be powerful knowledgeable and fast enough, and it should be adaptive in order to face all possible situations in the system.

The inputs to the production system are 14 components that are to be machined. Nine of the components are aimed towards assembly (a1, a2, a3, b1, b2, b3, c, d, e), and five components are just using the CMS for machining purposes (dummy1 to dummy5). Each component  $i$  has  $k_i$  processes to take, and each process can be performed on one of  $l_i$  machines from the set of five machines in the shop. A component can be processed more than once on the same machine, and the process times are  $T_{i,j}$  ( $i=1..k_i, j=1..l_i$ ) given in the database..

The control system main task is to route the parts dynamically according to the system state, part requirements and the assembly state.

### 5.1 ASSUMPTIONS

- I. The assembly times are not negligible in comparison with the production times. This assumption force

the production system to consider the feedback from the assembly, other wise the production should consider only the precedence of the parts.

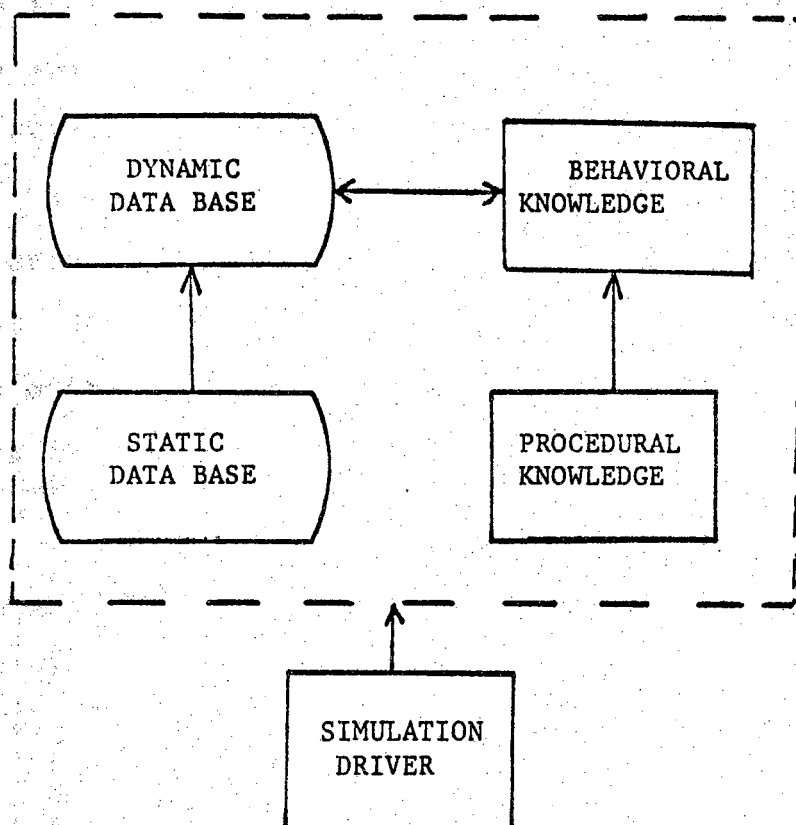
II. Parts in queues do not have any priority rule, and are processed in FIFO order.

III. The production system is balanced in the sense that the average processing time of the various alternatives is similar, and by chosing the route it is possible to control the arrival time of the finished component to the assembly station. If some of the parts are always late, there is no much need to dynamically control the system.

## 6. THE EXPERT SYSTEM STRUCTURE

The expert system is meant to introduce the context (or the environmental knowledge) into the decision making process. The objective of the proposed system is to utilize all the data available in a computerized manufacturing cell, create a good control mechanism to supervise the system and generate real time answers to problems arise during the system run time. The system basically is structured as shown below:

Figure 5. system structure



The control system interacts with the process controllers, gathers data from the shop floor and decide simple

decisions that reflects the automatic nature of the system. This system also consists of "algorithmic knowledge" and a simulation-driver.

Basically the expert system is in a production system form. A system of such type usually consists of three main parts:

1. Global database
2. Production rules.
3. Control (interpreter)

This gives the system the following advantages:

1. Modularity. It is easy to add rules, change them or delete part of the rules.
2. Uniformity in structure. All parts of the system are expressed in IF-THEN form.
3. Easy to understand.

The main disadvantage of the system is its inefficiency. It requires a lot of search to find the rules and process them.



## 6.1 KNOWLEDGE REPRESENTATION

In this system two types of knowledge exists: a production knowledge (rules), and procedural knowledge (which will be presented later on). The representation of the production type of knowledge is done in predicate form, using PROLOG. In PROLOG the knowledge in a form of clauses of first order predicate logic have three basic forms:

1. Facts. This type of clause is of the form

`p(a, b...).`

Example:

```
assembly(wheel, [quantity(a1, 3), quantity(a2, 1)]).  
part_process(a1, [drill1, mill3, bore9]).
```

2. Goals. In PROLOG this type of clause is of the form  
:-GOAL.

Example :

```
:- move_a_part(Machine, Process, Tool).
```

3. Procedures. This type of knowledge has the form

`A :- B, C, E.`

The ',' represent AND condition, and the ';' represent OR condition.

An example

```
simulate :- check_conditions,  
            update=current_time,  
            perform_event.
```

Although the PROLOG language claims to use first order

predicate logic, there are several crucial differences between the two. The simple clauses in PROLOG can be thought of as first order implications.  $P :- G$ , means  $G$  implies  $P$ . However the unification in PROLOG does not follow the rules of first order logic because PROLOG allows  $f(x,x)$  to match with  $f(y,g(y))$ , and so bind  $x$  to  $g(x)$  without noticing the circularity.

## 6.2 KNOWLEDGE ACQUISITION

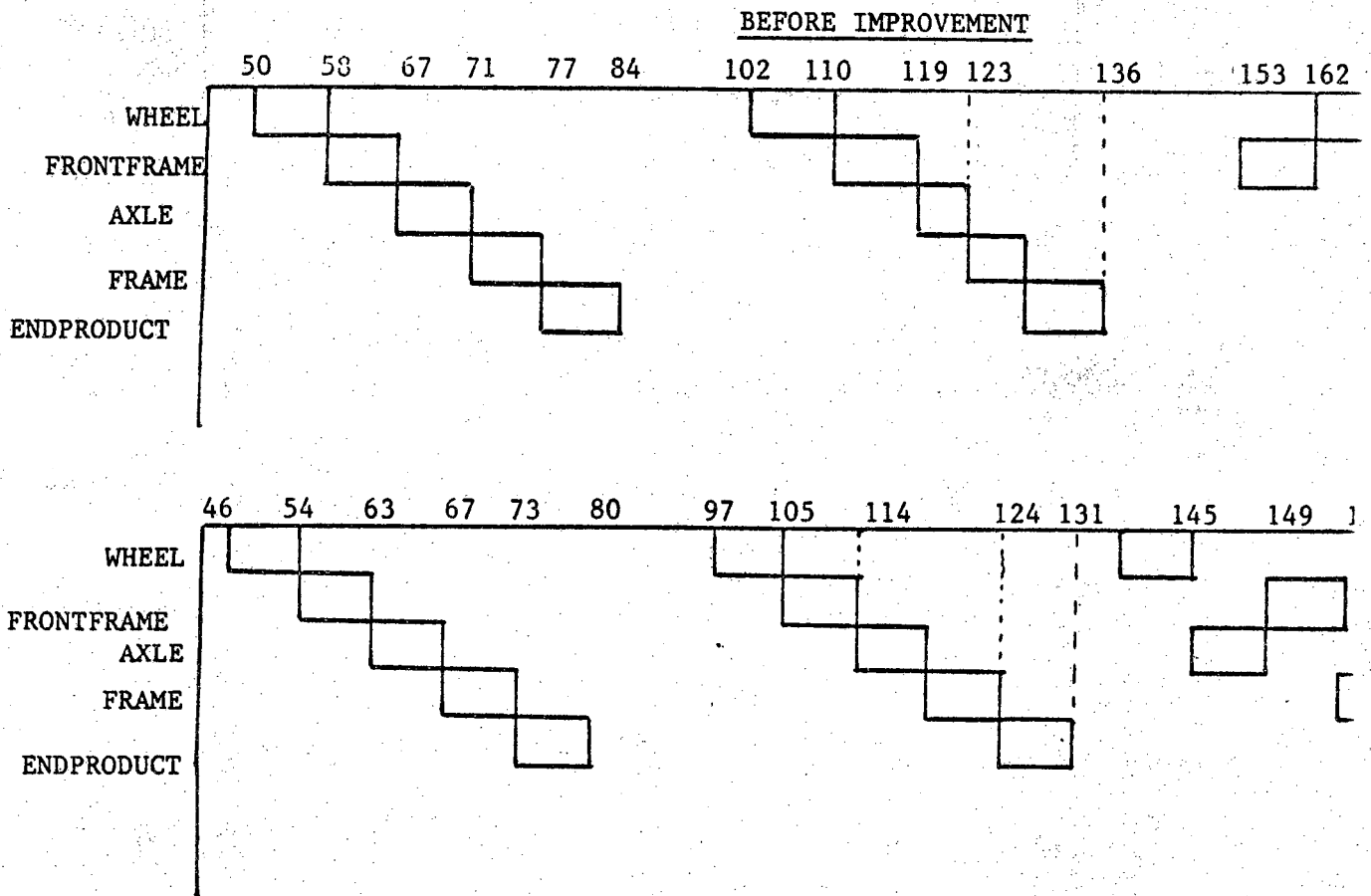
This system has two phases of acquisition of knowledge:

1. Initial knowledge acquisition. This step was taken when the system was built, and the knowledge about the desired behavior was embedded as a PROLOG predicates (IF-THEN rules).
2. Improvement phase. After the system was built several trial runs were made, and more knowledge about the system behavior was retrieved. This knowledge was added to the knowledge base.

As an example we can compare the performance of the assembly unit before the improvement and after. In this case after running the system, it was observed that parts in the production area do not consider similar parts that have been arrived to the assembly, and the production parts still use higher priority routes (shorter routes). Several more rules were introduced as

a result, and an improved behavior was observed. Comparison between the performance of the two systems can be seen in figure 6 . Although the utilization of the assembly station was only slightly improved, a major reduction of queue sizes resulted from this change.

Figure 6. comparison of system performance before and after improvement.

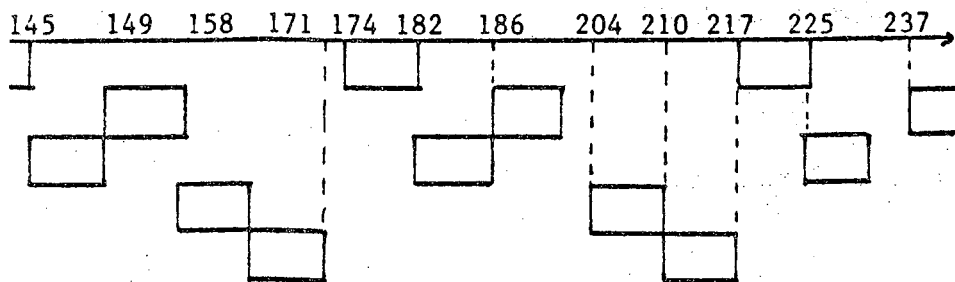
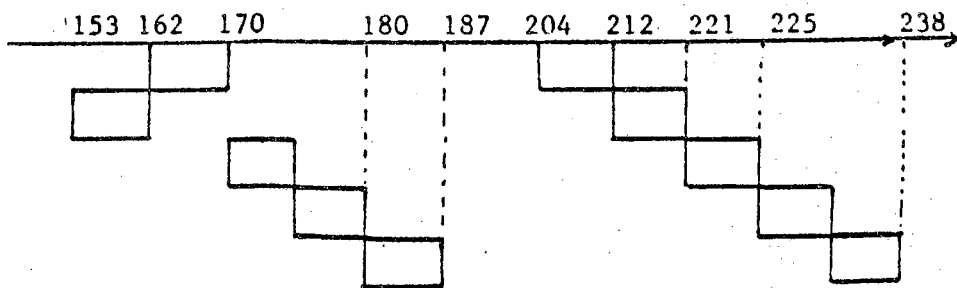


AFTER IMPROVEMENT

(continued)



(--- continued)



### 6.3 THE DATA BASE

The data base is the part of the expert system which stores the system states, the properties of the system components, and interacts with the system. This part does not include the knowledge base part. The data base for the system is composed of a static data base, and a dynamic one. The static data base contains information about the processes required, the part structure, the machines available and their capabilities. The dynamic data base contains data about queues in the system, parts' current process, the time a part is required and the assembly unit states.

The data base stores the information in predicates form, that is available in PROLOG.

#### EXAMPLE OF DATA ITEMS IN THE DATA BASE

```
/* PART CODE NUMBER part_type(code, part type) */  
part_type(1, a1).  
part_type(2, a2).  
  
/* PROCESS-MACHINE TABLE (PROCESS, [MACHINES LIST]). */  
pro_machine(1, [1, 2]).  
pro_machine(2, [3, 4, 5]).  
  
/* PART-PROCESS TABLE (PART, [ LIST OF PROCESSES ]). */
```

part\_pro(a1, [1, 8, 9]).

part\_pro(a2, [7, 5, 12]).

/\* PART-PROCESS-MACHINE-TIME TABLE \*/

/\* p\_t(PART NUM, [process(PROC. NUM, [time(MACHINE, TIME).. ])] ) \*/

part\_time(a1, [process(1, [time(1, 4), time(2, 6)]),

process(8, [time(3, 7), time(4, 9), time(5, 10)]),

process(9, [time(2, 4), time(3, 6), time(5, 8)])]).

/\* LIST OF THE ASSEMBLY TREE \*/

assembly(wheel, [quantity(a1, 1), quantity(a2, 2), quantity(a3, 3)]).

assembly(frontframe, [quantity(b1, 1), quantity(b2, 1), quantity(b3, 1)]).

/\* QUEUES' STATE : queue(MACH. NO, [q\_time(PART, TIME),... ])

Time is the (queue+process) time. It is the time a new  
part has to wait for process. \*/

queue(1, [q\_time(a1, 9), q\_time(d, 5)]).

queue(2, []).

/\* THE CURRENT PROCESS AND MACHINE: (Part, Process, Machine) \*/

current\_process(a1, 9, 1).

current\_process(a2, 7, 3).

### 6.3.1 INFORMATION RETRIEVAL

During run time of the system, the decisions and hypotheses are made based upon the system state at that

time. In order to have the desired information this system utilizes the query capability of the PROLOG language. The query process is based upon the predicate form of the PROLOG language and imitate the required predicate with a variable name for the required data item. This implies that the predicate form of the data base must be known in order to retrieve any desired item.

EXAMPLE

Data item in the database:

current\_process(a1, 9, 1).

Query:

current\_process(X, 9, Y).

Answer:

X = a1

Y = 1



#### 6.4 THE PRODUCTION RULES

This part contains the knowledge of the system (its expertise). There are three components of knowledge in this system: The first one consults with the data base and the system behavior knowledge and decides upon the system response to basic changes in its states. The behavioral knowledge of the system is represented in a combination of predicate logic and production rules, which is given by the PROLOG language.

Examples for the rules are:

```
IF      a part finishes a process
THEN    move the part to the next machine,
        supply the machine with the first part in queue,
        update the queues.
```

```
IF      part arrives.
THEN    find its first process,
        route it to a machine (found by the algorithm),
        add the part to the appropriate queue.
```

```
IF      a part is removed from the queue.
THEN    find the last part in the queue
        delete the part from the dynamic data base.
```

The representation of those conditions in PROLOG is:

```
part_arrival(Part) :- find_first_process(Part, M, A), !,
```

```
(big_number(B), A < B),  
add_to_queue(Part, M, A),  
create_current_process(Part, M),  
try_schedule_end_of_process(Part, M), !.
```

```
move_a_part(M, Part) :-    find_next_step(Part, X),  
                            not_last(X),  
                            find_next_process(Part, NewM, A),  
                            check_schedule(Part, NewM, A),  
                            remove_from_queue(M),  
                            schedule_next_part(M), !.
```

As it is seen from the example the objective of this part is to supply the system with the automatic nature it has. This part takes care of the part movements, activation of the various components of the system, and the bookkeeping required in the model.

The rules in this part can be divided into two main groups:

1. Finding to finding rules. These rules relate to events that occur in the system as the antecedent, and the hypothesis as the consequence. In this system the hypotheses are stated in action form, they are the actions that are assumed to be the right answers to the findings discovered.

#### EXAMPLE

```
IF    current_time(A),
      stop_time(B),
      A < B.
THEN  end_of_simulation_time(C),
      ACC.
```

2. Hypothesis to hypothesis rules. These are the more common rules in this system and they connect every hypothesis assumed to be true with all the other hypotheses that need to be checked (by performing an action).

EXAMPLE

```
IF    move_a_part(M, Part).
THEN  find_next_step(Part, X),
      not_last(X),
      find_next_process(Part, NewM, A)...
```

The second part of the knowledge is the algorithmic knowledge. This part solves the routing decision according to the current system states. The algorithm is dynamic and dependent upon the system current states as reflected in the dynamic data base, and not on mean off-line data.

The algorithm is composed of two parts:

- I. A production system written in PROLOG, that checks the logical conditions, retrieve the required data from the data base, and decides what is needed to

be solved.

II. A computational unit, written in PASCAL that manipulates the decisions made by the production system, and computes the cost of the various candidate solutions (next step in the algorithm). The algorithm is presented later on with an initialization procedure.

The third part of the knowledge is the simulation driver. A discrete simulation system is basically consists of three major parts:

1. The model of the system being simulated. This part defines the network structure of the system, the entities flow and the decisions required at each node. Usually this part is modeled by using a simulation language, or graphic symbols.
2. The data base of the simulation. This part is usually transparent to the user and keeps track of queues, entities, time of operations and collect the required statistics.
3. The event listing mechanism. This component chooses the next event to be processed, advances the simulation clock and motivate the entire simulation process.

The control system needs to have a simulation driver, because it is cheaper and more flexible to try its decision capability on a simulated manufacturing environment, instead a real one. In this way we ignore the need for the interface part of the system, but lose the flexible behavior of a real system. Since the system contains the data base and the behavioral knowledge for controlling the environment, the only component needed is the event file mechanism. This part recognizes three major types of events in the system:

1. Part arrivals. Initially all parts are introduced to the system in the same time, and later on a part that finishes production phase generates a request for the same type of part.
2. Process finishing. This event includes the decisions whether the part needs a route to the next process, needs to be sent to the assembly phase, and deals with the system ability to perform the decisions. When a machine becomes idle the first part in its queue is taken for production (FIFO order).
3. Assembly event. When there is any combination of parts in the assembly station that enables a subassembly or the end product to be assembled this event is performed.

This part is written in PROLOG , it is very simple and modular and enable the control system to react to simulated events, instead of real ones.

## 6.5 CONTROL STRATEGY

In this section several components of the control are described.

### RULE SELECTION

The main technique for rule selection is the matching technique. In PROLOG the matching is similar to unification in predicate logic with some variations.

### CONFLICT RESOLUTION

Sometimes several rules has the same L.H.S (therefor has the same name), and all of them can be triggered. In order to choose a specific rule to fire, the system (through PROLOG) select the rule according to their order in the database. In order to make a rule more favorable, it is possible to move the rule upwards in the list (or downwards for a lower likelihood rule).

Another strategy partially used is "context limiting" strategy. This approach checks the context of the rule, and only rules with the right context can be triggered. In this system the context is mentioned immediately in the R.H.S (Consequent) so the rule is first chosen by matching its name, and then by its order, and only then by the context.

### EXAMPLE

Rules for event that deals with assembly are in the form:

```
IF    the step is perform_event,
      the context is assembly.

THEN  change the status of the assembled part,
      add the part to the finished part storage,
      change status of the robot to "idle".
```

Another rule for event that deals with finishing a process is in the form:

```
IF    the step is perform_event,
      the context is part_finish.

THEN  identify the part,
      move the part.
```

ACTION In this system any rule that fires leads to an action. This action is a change in the system state as is reflected in the data base. Part that moves from a queue to a process, or enters a different queue, start assembly or finishes assembly, all of these changes are reflected in the database.

Changes in the database are utilized through predicates that are specified rules whose task is to create the desired change in the data base.

Examples for such rules are:

```
update_queue_time(T).
asserta(robot(free)).
update_event_list.
remove_from_queue(M).
add_to_queue(Part, M, A).
```



## 7. SUMMARY

This expert system tries to deal with a new domain: scheduling. In this field so far the expertise is limited and difficult to formulate. This system is used to schedule a specific environment in which a production system feeds an automated assembly station.

It is difficult to compare this system to others because today only one expert system for scheduling exists (ISIS), and there is no much information available about its scheduling process. As understood from the literature the ISIS system is used for higher level scheduling (daily schedule or even weekly or monthly), since it considers aggregate constraints. The presented system is used for real time scheduling and real time answers for scheduling problems, rather than long term schedule.

Computationally this system is expensive. Running on the VAX 11/780 it takes the computer about 9 sec. to load the six different files that compose the system:

File "po"	read time is 0.7 sec.
File "algo"	read time is 1.6 sec.
File "process"	read time is 1.75 sec.
File "states"	read time is 0.35 sec.
File "sim"	read time is 2.9 sec.
File "assembly"	read time is 1.35 sec.

To reach a decision about the part's route it takes the system from 0.1 sec to 0.25 sec, and a simulation of 240 min in simulation time takes about 7 minutes on the computer (in this period hundreds of decisions, data base changes and retrievals has to take place). However the slowest part in the system execution are the "system calls" that execute the compiled PASCAL code for the decision procedure.

The readability of the expert system is good because the predicates were given names that show their function and explain their task. A different user that runs the system can understand its decision process but needs some familiarity with PROLOG and its recursive nature.

The decision network for parts of the system is shown in the appendix where the branches are AND branches and are executed from left to right.

REFERENCES

- [1] Baker K.R., "Introduction to Sequencing and Scheduling", Wiley and Sons, 1974.
- [2] Bellman R., Esogbue A.O., Nabeshima I., "Mathematical Aspects Of Scheduling And Applications", Pergamon Press 1982.
- [3] Buchanan B.G., Duda R.D., "Principles of Rule-Based Expert Systems", Report no. HPP-82-14, Stanford university.
- [4] Bullers W.I., Nof S.Y., Whinston A.B., "Artificial Intelligence In Manufacturing Planning And Control", AIIE Trans Dec, 1980.
- [5] Buzacott J.A., "The production capacity of job shops with limited storage space", Int J. Prod. Res. vol. 14, no 5, 1976.
- [6] Buzacott J.A., Shanthikumar J.G., "Models for Understanding Flexible Manufacturing Systems", AIIE Trans. 12(4), Dec 1980, pp. 339-350.
- [7] Buzacott J.A., "Optimal operating rules for automated manufacturing systems", IEEE Trans. Automat. Contr. vol. 27, no 1, 1982.
- [8] Buzen J.P., "Computational Algorithms For Closed Queueing Networks with Exponential Servers", Comm.

of ACM, vol 16, no. 9, 1973, pp. 527-531.

- [9] Conner J.L., "A Heuristic Method for Solving Job Shop Sequencing Problem", Production and Inventory Mgt. 13, 1, pp. 54-68, 1972.
- [10] Davis R., King J., "An overview of production systems", Machine Intelligence vol 8, pp 300-332.
- [11] Day J.E. and Hottenstein M.P., "Review of Sequencing Research", Naval Research Logistics Quarterly 1970.
- [12] Denning P.J., Buzen J.P., "The Operational Analysis of Queueing Networks Models", Computing Surveys vol. 10, no. 3, 1978.
- [13] Ferguson R.L., Curtis H.J., "A Computer Aided Decision System", Mgt. Science 15, 10, pp. 550-561, June 1969.
- [14] Fox M.S., Allen B., Strohm G., "Job-Shop Scheduling: An Investigation in Constraint-Directed Reasoning", Proc. NCAI 1982, Pittsburg, PA, pp. 155-158.
- [15] Fox M.S., "Constraint Directed Search: A Case Study of Job-Shop Scheduling", Ph.D Thesis, Carnegie-Mellon University 1983.
- [16] Futo I., Szeredi J., "System Simulation and Cooperative Problem Solving on a PROLOG Basis", in

"Implementations of PROLOG" edited by Campbell J. A., Ellis Horwood, 1984.

- [17] Godin V. B., "Interactive Scheduling: Historical Survey and State of the Art", AIIE Trans. Sep 1978.
- [18] Gordon W. J., Newell G. F., "Closed Queueing Networks with Exponential Servers", Operations Research vol 15, no. 2, Apr 1967, pp. 244-265.
- [19] Hart P. E., Nilsson N. J., Raphael B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Trans. systems science and cybernetics vol ssc-4, no. 2, July 1968.
- [20] Hatvany J., Lettner F. J., "The Efficient Use Of Deficient Knowledge", Annals of the CIRP, vol 32/1/1983.
- [21] Hayes J. E., Michie D. "Intelligent Systems", Ellis Horwood, 1983.
- [22] Hilderbrant R. R., "Scheduling Flexible Machining Systems Using Mean Value Analysis", Proc. of IEEE conference on Decision and Control, Albuquerque, N. Mexico, 1980.
- [23] Jackson J. R., "Jobshop like Queueing Systems", Management Science vol 10, no. 1, Oct 1963,

pp. 131-142.

- [24] Kimemia J., Gershwin S.B., "An algorithm for the computer control of a flexible manufacturing system", IIE Trans. dec 1983.
- [25] King J.R., Spachis A.S., "Scheduling: Bibliography and Review", I.J. of Physical Distribution and Material Management, vol. 10 (3).
- [26] Klahr P., Faught W.S., "Knowledge Based Simulation", in proceedings of the first annual national conference on AI, Stanford 1980.
- [27] Kusiak A., "Flexible manufacturing systems: a structural approach", working paper no. 4/84, university of Nova Scotia.
- [28] McDermott D., "The Prolog Phenomenon", in SIGART newsletter, no. 72, July 1980, pp. 16-20.
- [29] McLean C.R., Bloom H.M., and T.H. Hopp, "The virtual manufacturing cell", NBS report.
- [30] Mizoguchi F., "PROLOG Based Expert System", in "New Generation Computing", no. 1, 1983, pp. 99-104.
- [31] Nau D.S., "Expert computer systems", Computer, Feb 1983.
- [32] Nilsson N.J., "Principles of Artificial Intelli-

gence", Tioga Pub., 1980.

[33] Nof S. Y., "An Expert System For Planning/Replanning Programmable Production Facilities", Proc ICPR Aug 1983, Windsor Canada.

[34] Pohl I., "First Results on the Effect of Error in Heuristic Search", in Machine Intelligence, vol 5, 1970, pp 219-236.

[35] Pohl I., "Practical and Theoretical Considerations in Heuristic Search Algorithms", in Machine Intelligence, 9, 1977, pp 55-72.

[36] Rauch H., Firschein D., Perkins W.A. and Pecora V. J., "An Expert System for Tactical Data Fusion", IEEE conf. on circuits, systems and computers, 1982.

[37] Simon H. A., Newell A., "Heuristic Problem Solving: The Next Advance in Operations Research", Operations Research 6, 1, pp. 1-10, Jan. 1958.

[38] Solberg J. J., "Quantitative Design Tools for Computerized Manufacturing Systems", Proc. Sixth North-American Metalworking Research Committee, Gainesville, Florida, Apr 1978.

[39] Steckle K. E., "The optimal planning of Computerized Manufacturing Systems", Ph.D Thesis, Purdue

University, 1981.

- [40] Stefik M. et al., "The organization of expert systems", AI, 18 (1982) pp. 135-173.
  
- [41] Suardo G.M.S., "Workload optimization in a FMS Modeled as a Closed Network of Queues", Fiat Research center, CIRP Annalen 28(1), pp. 381-383.
  
- [42] Suresh J.K., "A Simulation Based Scheduling and Management Information System for a Machine Shop", Interfaces 6,1 part 2, pp.81-96, Nov. 1975.
  
- [43] Warren D., Pereira F., "PROLOG- The Language and its Implementation Compared with LISP", SIGART Newsletter, no.64, Aug 1977.



```

*****
*
*   A P P E N D I X   *
*
*****

```

B. THE ALGORITHM: EVALUATION OF PARTIAL SOLUTIONS

The scheduling algorithm has to search through many possible actions that the scheduler can take. These alternatives have to be evaluated and the best one is to be chosen. These possible assignments of jobs to machines depends upon the availability of the machines, queues that each machine has, type of process to be done next, capability of the machines to perform the task and other criteria. In order to evaluate the proposed route the scheduler needs to evaluate the route according to some measure of performance. The route decision is a tree like decision procedure: first the next process has to be scheduled than the next one (which depends upon it), and so on. It is desired to have an evaluation function of partial solutions in order to reduce the number of possible solutions generated.

B.1 AN ALGORITHM FOR HEURISTIC SEARCH

In case of heuristic search there are several known algorithms, of which the best known is algorithm A\* (Nilsson 1980, Hart Nilsson and Raphael 1967) or HPA (Pohl 1970, 1977). This algorithm makes use of an evaluation function for partial solutions of the form:

$$f(x) = (1-w)g(x) + wh(x).$$

where s = start node, t = terminal node.

g(x) = The 'distance' from s to x as found in the search.

$h(x)$  = The estimated 'distance' from  $x$  to  $t$ .

$w$  = weighting factor.

This function has some conditions to follow in order to get desirable properties. The requirements are:

I.  $h(n) \leq h^*(n)$ .

where  $h^*(n)$  is the cost of the minimal path from any  $n$  to  $t$ .

II. Consistency:  $(h(x) - h(y)) \leq c(x, y)$  for all  $x, y$ . Here  $c(x, y)$  is the real distance from  $x$  to  $y$ .

III.  $0.5 \leq w \leq 1$ .

The properties of this search are:

I. Admissibility. The algorithm always terminate in an optimal path from  $s$  to  $t$ .

II. Optimality. The search includes the fewest nodes in finding a solution path (Pohl 1970).

III. The more informed the algorithm is the shortest the search is. By information one means the accurate estimate of  $h(n)$ .

IV. The algorithm can use dynamic weighting  $w(x)$  that changes according to the distance of  $x$  from  $s$ .

## 8.2 SEARCH PROCEDURE FOR THE ROUTING PROBLEM

The routing problem differs from the general search problem by several important properties: First only the initial state is known. The goal node is not known, therefore the solution to the search is not only the path from  $s$  to  $t$  but also the terminal node itself. A second major difference is that the routing problem with the proposed objective function is not a consistent problem. It means that  $h(x) - h(y) \neq c(x, y)$ , a situation caused by the fact that some of the arcs have a 'negative' cost which reduce the objective function when added. Another difference is that the routing problem is dynamic problem which means that the same subproblem can have various values under different conditions. One of the results is that  $h(n)$  cannot be evaluated in advance. These properties do not allow the use of algorithm  $A^*$  as it is and some modifications must be done.

On the other hand since the search algorithm can be very 'informed' it is desired to use all the information available to reduce the search space. The information can aid in considering the alternatives in various times and states (which imply different values for the evaluation function even for the same branch).

### 8.3 THE SEARCH PROCEDURE

In order to evaluate various solutions the first step is to solve arbitrarily the routing problem, then to evaluate this solution according to the evaluation function and improve the solution by considering only the candidate solutions that can improve the function. By identifying undesired solutions the candidate list is reduced and an exhausting search is avoided. The solution estimation is performed according to the knowledge available about the system and the consequences of the various partial solutions.

The evaluation function is:

$$f(x) = MxIt + \sum_j w_j$$

The  $w$  is the waiting time of the part being routed and  $j$  belongs to the set of available machines (including the assembly station).

The 'It' is the idle time of the assembly station.

In order to discuss the algorithm more concretely the use of graph model is made for the heuristic search and a program which can search problems represented in the model.

The problem space is the finite tree  $T$ .

$T = (X, E)$  is the feasible process assignment tree.

$X = \{x_1, x_2, \dots\}$  the set of nodes.

$E = \{e_1, e_2, \dots\}$  the set of edges such that  $e_j =$

$(x_i, x_j)$ ,  $x_i, x_j \in X$  and  $x_j \in \Gamma(x_i)$ .

$\Gamma$  is the successor mapping and if  $x = \Gamma(y)$  then  $y = \Gamma^{-1}(x)$ .

It is important to notice the special structure of the tree. Since the successor edges do not depend upon the predecessor it means that  $\Gamma(x_i) = E(i) = \Gamma(x_j)$ , where  $x_i, x_j$  are in the same level in the tree.

Each edge  $i$  in the tree (connects  $x_h$  to  $x_i$ ) have a finite cost associated with it and this cost can be represented as a tuple  $(A_i, w_i)$ .

$w_i$  is the waiting time of the part on the machine.

$A_i$  is the total time spent in the machine ( $A_i = p_i + w_i$ , and  $P_i$  is the process time).

A solution is called optimal if it has the least value of  $f(x)$  (least costly path).

Other symbols are:

$C: E \rightarrow \mathbb{R}^2$ , the cost of the edges.

$m(s, t) = (s=x_1, x_2, \dots, x_k=t)$  the path from  $s$  to  $t$ .

$c(m(s, t)) = M(I_{t+}) + (I_{t-}) + \sum_j w_j$ ,  $j \in m(s, t)$ .

$I_{t+}$  is  $\sum A_i - E_t$ , where  $\sum A_i \geq E_t$  (the idle time of the assembly station).

$I_{t-}$  is  $|\sum A_i - E_t|$  where  $E_t > \sum A_i$  (the waiting time for assembly).

$E_t$  is the time the part is required for the assembly (due date).

#### 8.4 THE ALGORITHM

1. Start with  $x_i = x_1 = s$ , calculate  $x_j = \Gamma(x_i)$  (for process  $j$  that follows  $i$ ), such that  $A_j = \text{MIN} \{A_k \mid k \in E_i\}$ .

Repeat step 1 until  $\Gamma(x) = 0$ . (no more processes to take).

Calculate  $f(x)$  for the path found.

2. If  $It+ \geq 0$  exit. The path is optimal for  $It+ > 0$ .

Else start with  $E(i) = E(1)$ .

- i. Remove from  $E(i)$  all arcs  $e_j$  for which  $w_j > w_i$  ( $w_i$  belongs to  $e_i$  found in 1).

- ii. Replace  $e_i$  with  $e_j$  such that  $(A_j - A_i) \leq |It|$  and  $w_j = \text{MIN} \{w_k \mid k \in E(i)\}$ .

Calculate  $f(x)$ .

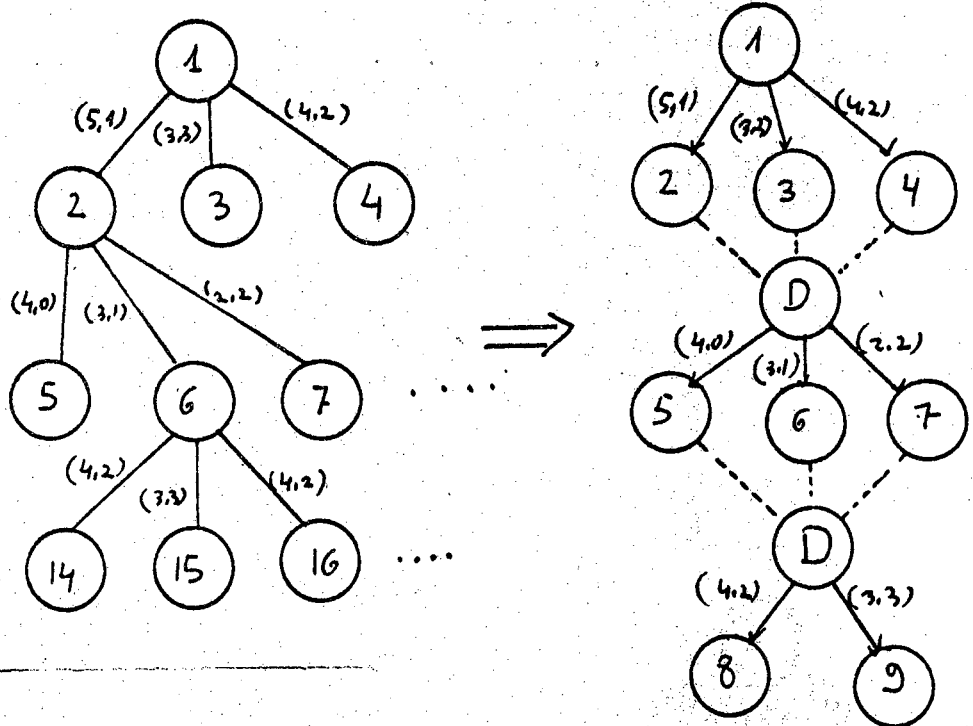
- iii. Remove  $e_i$  from  $E_i$ .

- iv. Continue until  $E_i \wedge e_i = 0$ .

3. Calculate  $\Gamma(x_i)$ . If  $\Gamma(x_i) \neq 0$ , go back to 2.

#### EXAMPLE

Figure 7. example



Given:  $E_t = 10$ .

1. Calculate the initial path:

$x_1 = 1$ .

$x_2 = 3$ .

$x_3 = 7$ .

$x_4 = 9$ .

The path is  $\{(1,3), (3,7), (7,9)\}$

$f(x) = M(0) + 2 + 8 = 10$ .

2. i.  $E(1) = \{(1,2), (1,3), (1,4)\}$

There are no edges to remove ( $w_i = \text{MAX } \{w_j\}$ ).

ii.  $|It| = 2$ .

$\text{MIN } \{w_j\} = 1$  for edge  $(1,2)$ .

Replace  $(1,3)$  with  $(1,2)$  with cost =  $(5,1)$ .

$E(1) = \{(1,2), (1,4)\}$ .

$f(x) = 6$  and  $It=0$ .

Since  $I_t=0$   $e_4$  can be removed (cannot replace  
e2)

In step 2 we exit with the final path:

$\{(1,2), (2,7), (7,9)\}$ . The cost is 6.



## 8.5 THE INITIALIZATION ALGORITHM

### Symbols

Component: An elementary part (not subassembly).

$atime_j$ : the assembly time of part (component)  $j$ .

expected time $_j$ : The time part  $j$  is required at the assembly.

$P(j)$ : The successor mapping (finds immediate sons of  $j$ ).

Input: assembly tree  $T(A, V)$ , assembly times for part  $j$ :  $atime_j$  for all  $j \in V$ .

Output: list of expected times for all components and parts in  $T$ ; expected time $_j$ , for all  $j \in V$ .

### THE ALGORITHM

1. Find the set of all the minimal subtrees  $[I, J, \dots]$  that are composed only of components.

Set  $Et = 0$ .

2. Until all subtrees are covered do:

- i. All components  $j \in I$  are assigned expected time $_j = Et$ .
- ii. expected time $_I = Et$ .
- iii.  $Et = Et + atime_I$ .

- 3.

- i.  $Ct = 0$ .
- ii. Find  $A$ , such that  $j \in A$  and expected time <sub>$j$</sub>  =  $Ct$ . (find set of all parts and components with expected time =  $Ct$ ). If  $A$  contains component  $j$  then find  $k = \Gamma^{-1}(j)$  (the root of  $j$ ), and  $l = \Gamma^{-1}(k)$ .

Else (only part in  $A$ )  $l = \Gamma^{-1}(j)$ .

- iii. If node  $l$  has no expected time assigned to it, and all its predecessors have then:

For every component  $j$  arrive to  $l$   
assign: expected time <sub>$j$</sub>  =  $Et$ .

$$\text{expected time}_1 = Et.$$

$$Et = Et + \text{atime}_1.$$

$$Ct = Ct + \text{atime}_\Gamma(l)$$

- iv. If  $l \neq$  endproduct (the root of the tree) go to ii.

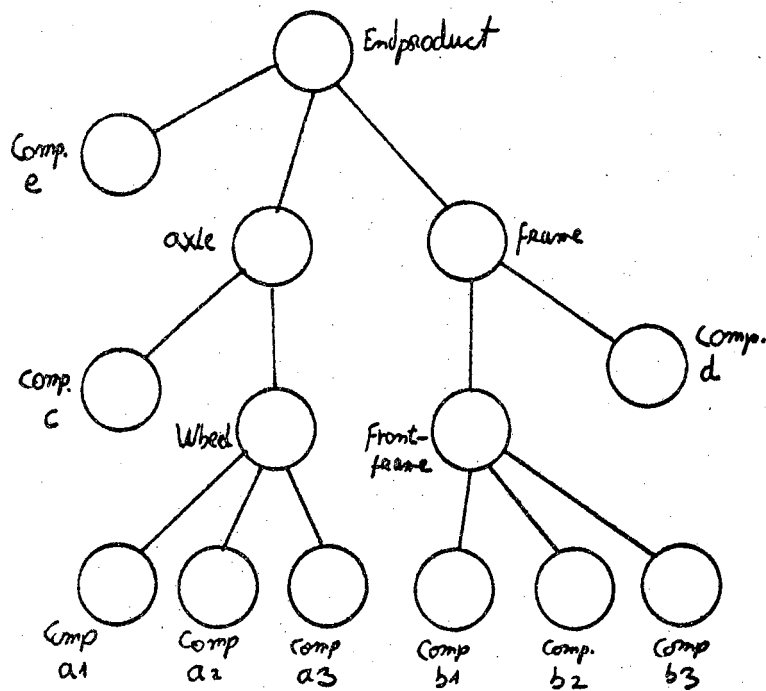
**EXAMPLE**

**Assembly Times**

---

Part	Time
Wheel	8
Frontframe	9
Axle	4
Frame	6
Endproduct	7

---



Step 1: The parts are wheel composed of a1, a2, a3.

frontframe (also ff) composed of b1, b2, b3.

Step 2: a1, a2, a3 have expected time = 0.

expected time<sub>wheel</sub> = 0.

Et = 8.

b1, b2, b3 have expected time = 8.

expected time<sub>ff</sub> = 8.

Et = 17.

Step 3:

i. Ct = 0.

A = {a1, a2, a3} with expected time = 0.

A contains only components so  $\Gamma^{-1}(A) = \text{wheel}$ .

l =  $\Gamma^{-1}(\text{wheel}) = \text{axle}$ .

part c which arrives to axle gets expected time<sub>c</sub> = 17.

c is the only component so Et = 17+8 = 25.

ct = 0 + atime<sub>(wheel)</sub> = 8.

ii. ct = 8, A = {b1, b2, b3}, k =  $\Gamma^{-1}(A) = \text{ff}$ .

l =  $\Gamma^{-1}(k) = \text{frame (or f)}$ .

part d gets expected time<sub>d</sub> = 25.

Et = 25+3 = 28. Ct = 8+9 = 17.

iii. A = {c}, k = axle, l = endproduct.

l gets expected time = 28.

Et = 28+6 = 34.

Ct = 17 + atime<sub>axle</sub> = 25.

Since l is endproduct we are done.

current\_time 0 part b1 - 92 -

G THE OPTIMAL ROUTE IS G

CHOOSE MACHINE 4

CHOOSE MACHINE 4

CHOOSE MACHINE 2

THE OPTIMAL ROUTE COST IS: 1303

current\_time 0 part b2

G THE OPTIMAL ROUTE IS G

CHOOSE MACHINE 5

CHOOSE MACHINE 1

CHOOSE MACHINE 5

CHOOSE MACHINE 5

THE OPTIMAL ROUTE COST IS: 2204

current\_time 0 part b3

G THE OPTIMAL ROUTE IS G

CHOOSE MACHINE 3

CHOOSE MACHINE 5

CHOOSE MACHINE 2

CHOOSE MACHINE 1

THE OPTIMAL ROUTE COST IS: 2117

current\_time 0 part c

G THE RESULT OF THE ALGORITHM G

THE EXPECTED TIME IS : 17

THE INITIAL ARRIVAL TIME IS: 17

THE ROUTE FOR THE PART IS:

machine no:	2	10
machine no:	1	7

267 statements executed in 0000.150 seconds cpu time.

THE ROUTE COST IS: 7

SYSTEM OUTPUT

```
current_time      event_list
0      [(3, part_finish, 2, a2), (4, part_finish, 1, a1), (5, part_finish, 5, b2),
        (5, part_finish, 3, a3), (9, part_finish, 4, b1)]
queue(2, [q_time(dummy5, 22), q_time(dummy3, 17), q_time(c, 10), q_time(a2, 3)]).
queue(1, [q_time(dummy4, 19), q_time(dummy1, 14), q_time(d, 7), q_time(a1, 4)]).
queue(3, [q_time(dummy2, 15), q_time(e, 8), q_time(b3, 6), q_time(a3, 5)]).
queue(5, [q_time(b2, 5)]).
queue(4, [q_time(b1, 9)]).
queue(assembly, '[]').
```

```
current_process(dummy5, 96, 2).
current_process(dummy4, 99, 1).
current_process(dummy3, 93, 2).
current_process(dummy2, 92, 3).
current_process(dummy1, 90, 1).
current_process(e, 2, 3).
current_process(d, 1, 1).
current_process(c, 6, 2).
current_process(b3, 2, 3).
current_process(b2, 11, 5).
current_process(b1, 6, 4).
current_process(a3, 10, 3).
current_process(a2, 7, 2).
current_process(a1, 1, 1).
```

```
finished_storage : [ ] current_time      event_list
3      [(4, part_finish, 1, a1), (5, part_finish, 5, b2), (5, part_finish, 3, a3),
        (9, part_finish, 4, b1), (10, part_finish, 2, c)]
queue(2, [q_time(dummy5, 19), q_time(dummy3, 14), q_time(c, 7)]).
queue(5, [q_time(a2, 11), q_time(b2, 2)]).
queue(assembly, '[]').
queue(4, [q_time(b1, 6)]).
queue(3, [q_time(dummy2, 12), q_time(e, 5), q_time(b3, 3), q_time(a3, 2)]).
queue(1, [q_time(dummy4, 16), q_time(dummy1, 11), q_time(d, 4), q_time(a1, 1)]).
```

```
current_process(a2, 5, 5).
current_process(dummy5, 96, 2).
current_process(dummy4, 99, 1).
current_process(dummy3, 93, 2).
current_process(dummy2, 92, 3).
current_process(dummy1, 90, 1).
current_process(e, 2, 3).
current_process(d, 1, 1).
current_process(c, 6, 2).
current_process(b3, 2, 3).
current_process(b2, 11, 5).
current_process(b1, 6, 4).
current_process(a3, 10, 3).
current_process(a1, 1, 1).
```

```
finished_storage : [ ] current_time      event_list
4      [(5, part_finish, 5, b2), (5, part_finish, 3, a3), (7, part_finish, 1, d),
        (9, part_finish, 4, b1), (10, part_finish, 2, c)]
```

```
/* DEFINITION OF AN INFINITE NUMBER */  
big_number(1000).
```

```
/* NUMBER OF COMPONENTS IN THE SYSTEM */  
part_types(9).
```

```
/* PART CODE NUMBER part_type(code, part type) */  
part_type(1, a1).  
part_type(2, a2).  
part_type(3, a3).  
part_type(4, b1).  
part_type(5, b2).  
part_type(6, b3).  
part_type(7, c).  
part_type(8, d).  
part_type(9, e).  
part_type(10, dummy1).  
part_type(11, dummy2).  
part_type(12, dummy3).  
part_type(13, dummy4).  
part_type(14, dummy5).
```

```
/* PROCESS-MACHINE TABLE (PROCESS, [MACHINES LIST]). */
```

```
pro_machine(1, [1, 2]).  
pro_machine(2, [3, 4, 5]).  
pro_machine(3, [1, 3, 4]).  
pro_machine(4, [4, 5]).  
pro_machine(5, [1, 4, 5]).  
pro_machine(6, [2, 3, 4]).  
pro_machine(7, [1, 2, 4]).  
pro_machine(8, [3, 4, 5]).  
pro_machine(9, [2, 3, 5]).  
pro_machine(10, [1, 2, 3, 4]).  
pro_machine(11, [1, 3, 4, 5]).  
pro_machine(12, [1, 2, 3, 4, 5]).  
pro_machine(13, [2, 3, 4, 5]).
```

```
/* PART-PROCESS TABLE (PART, [ LIST OF PROCESSES ]). */
```

```
part_pro(a1, [1, 8, 9]).  
part_pro(a2, [7, 5, 12]).  
part_pro(a3, [10, 5, 7, 2]).  
part_pro(b1, [6, 8, 12]).  
part_pro(b2, [11, 3, 5, 8]).  
part_pro(b3, [2, 4, 13, 7]).  
part_pro(c, [6, 12]).  
part_pro(d, [1, 5, 7, 9]).  
part_pro(e, [2, 3, 10, 11]).  
part_pro(dummy1, [90, 99, 98]).  
part_pro(dummy2, [92, 94, 95]).  
part_pro(dummy3, [93, 94, 98]).  
part_pro(dummy4, [99, 92, 97]).  
part_pro(dummy5, [96, 99, 92]).
```

```
/* PART-PROCESS-MACHINE-TIME TABLE */
```

```
/* p_t(PART NUM, [process(PROC. NUM, [time(MACHINE, TIME).. ])] ) */
```

```
part_time(a1, [process(1, [time(1, 4), time(2, 6)]),  
              process(8, [time(3, 7), time(4, 9), time(5, 10)])],
```

```
        process(9, [time(2, 4), time(3, 6), time(5, 8)]))].
part_time(a2, [process(7, [time(1, 2), time(2, 3), time(4, 5)]),
              process(5, [time(1, 7), time(4, 9), time(5, 9)]),
              process(12, [time(1, 2), time(2, 3), time(3, 4),
                           time(4, 5), time(5, 6)]))].
part_time(a3, [process(10, [time(1, 2), time(2, 4), time(3, 5), time(4, 7)]),
              process(5, [time(1, 3), time(4, 5), time(5, 5)]),
              process(7, [time(1, 5), time(2, 6), time(4, 9)]),
              process(2, [time(3, 6), time(4, 6), time(5, 6)]))].
part_time(b1, [process(6, [time(2, 7), time(3, 9), time(4, 9)]),
              process(8, [time(3, 2), time(4, 4), time(5, 6)]),
              process(12, [time(1, 5), time(2, 5), time(3, 8),
                           time(4, 10), time(5, 10)]))].
part_time(b2, [process(11, [time(1, 2), time(3, 2), time(4, 5), time(5, 5)]),
              process(3, [time(1, 4), time(3, 4), time(4, 5)]),
              process(5, [time(1, 5), time(4, 6), time(5, 8)]),
              process(8, [time(3, 5), time(4, 8), time(5, 9)]))].
part_time(b3, [process(2, [time(3, 1), time(4, 2), time(5, 5)]),
              process(4, [time(4, 7), time(5, 7)]),
              process(13, [time(2, 2), time(3, 5), time(4, 6), time(5, 6)]),
              process(7, [time(1, 2), time(2, 5), time(4, 9)]))].
part_time(c, [process(6, [time(2, 7), time(3, 8), time(4, 10)]),
              process(12, [time(1, 3), time(2, 5), time(3, 7),
                           time(4, 10), time(5, 10)]))].
part_time(d, [process(1, [time(1, 3), time(2, 6)]),
              process(5, [time(1, 3), time(4, 6), time(5, 7)]),
              process(7, [time(1, 4), time(2, 5), time(4, 8)]),
              process(9, [time(2, 2), time(3, 3), time(5, 6)]))].
part_time(e, [process(2, [time(3, 2), time(4, 4), time(5, 6)]),
              process(3, [time(1, 4), time(3, 6), time(4, 7)]),
              process(10, [time(1, 2), time(2, 3), time(3, 5), time(4, 5)]),
              process(11, [time(1, 6), time(3, 10), time(4, 10), time(5, 10)]))].

part_time(dummy1, [process(90, [time(1, 7)]),
                  process(99, [time(3, 6)]), process(98, [time(5, 5)]))].
part_time(dummy2, [process(92, [time(3, 7)]),
                  process(94, [time(4, 3)]), process(95, [time(1, 5)]))].
part_time(dummy3, [process(93, [time(2, 7)]),
                  process(94, [time(4, 6)]), process(98, [time(3, 6)]))].
part_time(dummy4, [process(99, [time(1, 5)]),
                  process(92, [time(3, 6)]), process(97, [time(2, 5)]))].
part_time(dummy5, [process(96, [time(2, 5)]),
                  process(99, [time(3, 6)]), process(92, [time(5, 5)]))].

/* BUFFER SIZES */
buffer_size(1, 6).
buffer_size(2, 6).
buffer_size(3, 6).
buffer_size(4, 6).
buffer_size(5, 6).
buffer_size(assembly, 27).
```



THE DYNAMIC DATA BASE

```
/* EXPECTED TIME OF PART X */  
/* The program computes these values */
```

```
/* QUEUES' STATE : queue(MACH. NO, [q_time(PART, TIME), ...])  
Time is the (queue+process) time. It is the time a new  
part has to wait for process. */
```

```
queue(1, []).  
queue(2, []).  
queue(3, []).  
queue(4, []).  
queue(5, []).
```

```
/* TOOL AVAILABLE : tool(MACH. NO, [TOOL NO. ...]) */
```

```
tool(1, [1, 2, 4, 6, 8, 12]).  
tool(2, [2, 3, 4, 5, 7, 9, 10]).  
tool(3, [1, 3, 5, 6, 7, 9]).  
tool(4, [10, 11, 12, 13]).  
tool(5, [1, 6, 8, 10, 12]).
```

```
/* MACHINE STATUS : status(MACH. NO, FAIL/WORK). */
```

```
status(1, 1).  
status(2, 1).  
status(3, 1).  
status(4, 0).  
status(5, 1).
```

```
/* THE CURRENT PROCESS AND MACHINE: (Part, Process, Machine) */
```

```
/*  
current_process(a1, 9, 1).  
current_process(a2, 7, 3).  
current_process(b1, 8, 4).  
current_process(b2, 11, 2).  
current_process(b3, 1, x).  
current_process(c, X, x).  
current_process(d, X, u).  
current_process(e, X, m).  
*/
```

```
/* REPAIR TIMES */  
repair_time(M, T).
```

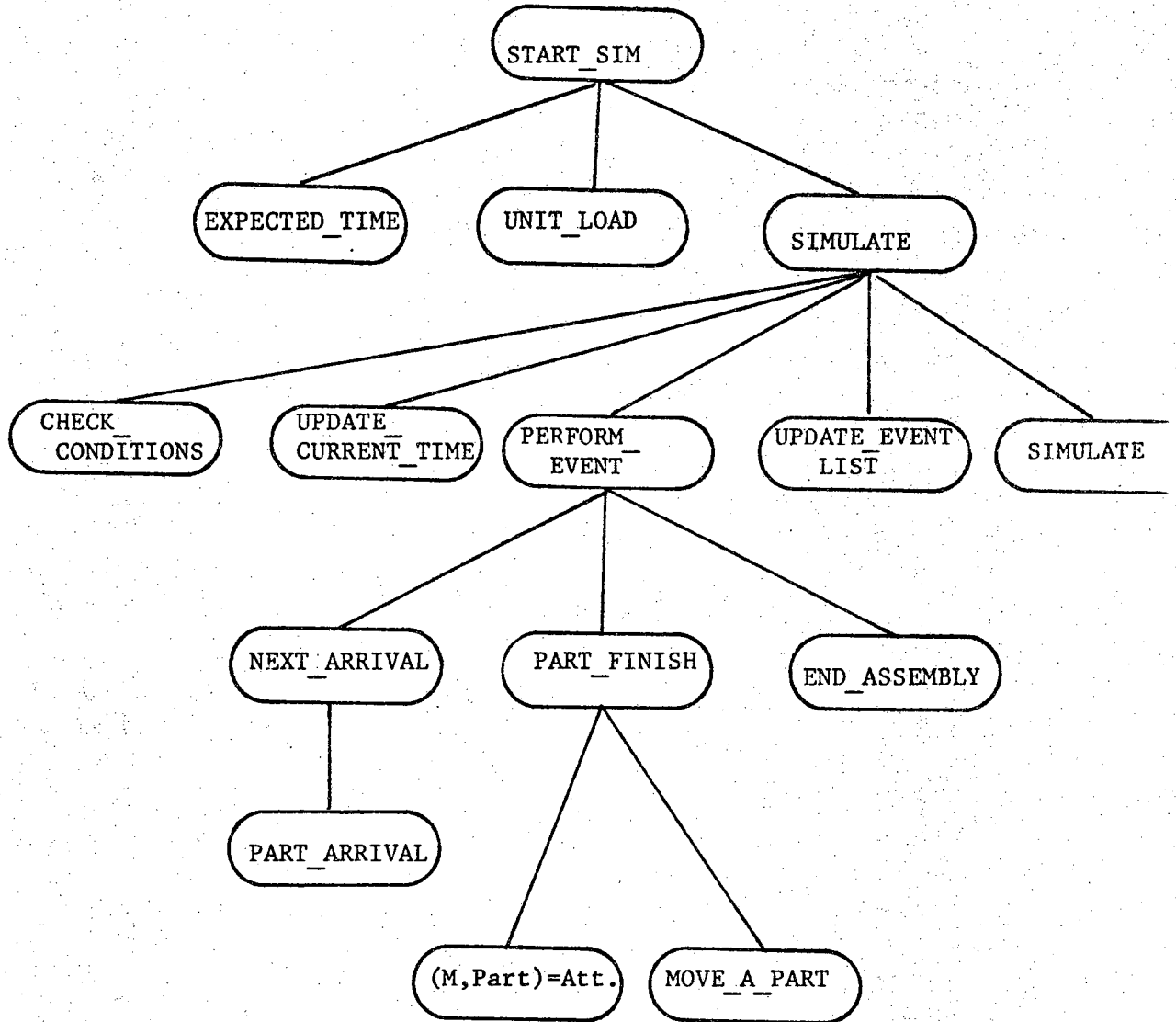
```
/* CENTRAL STORAGE CONTENTS */  
central_storage([]).
```

```
/* ASSEMBLY STATION CONTENTS */  
queue(assembly, []).
```

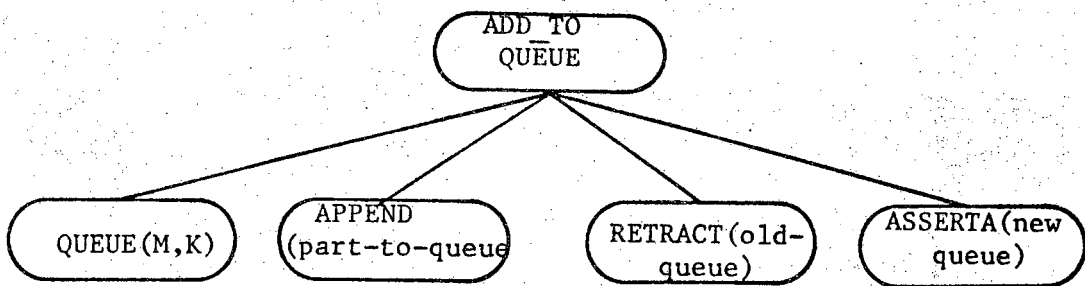
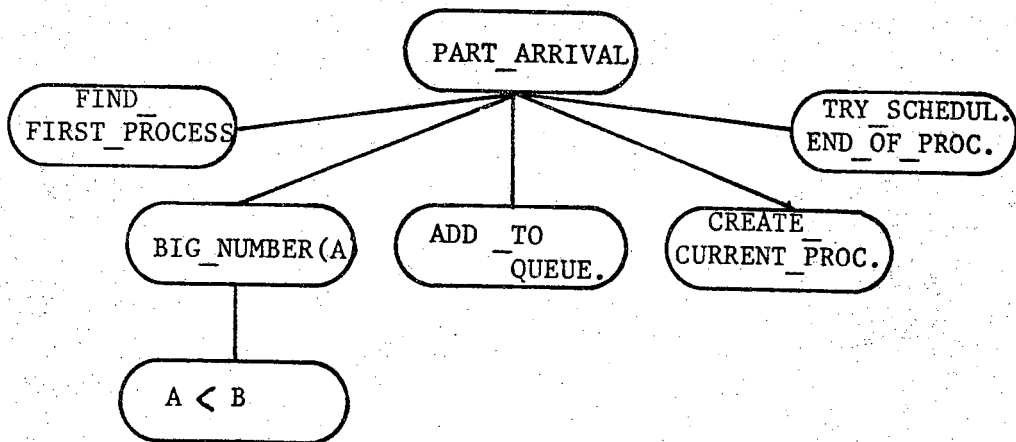
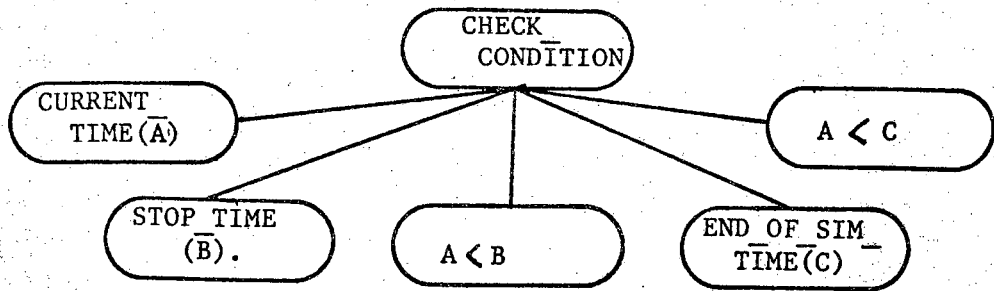
```
/* FINISHED PARTS STORAGE */  
finished_storage([]).
```

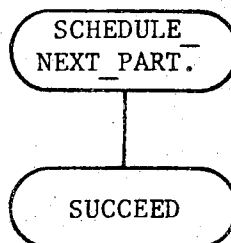
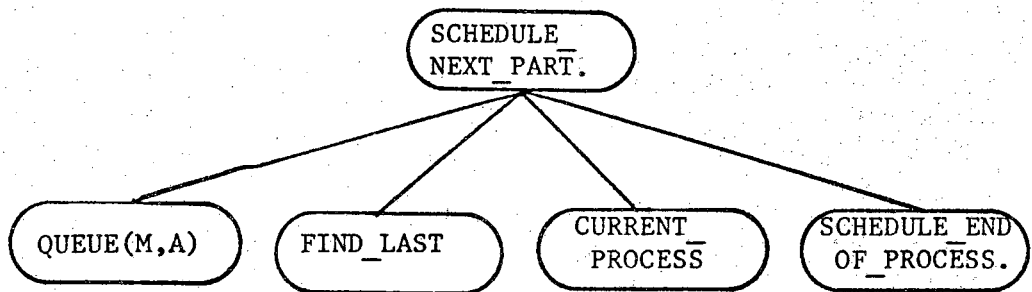
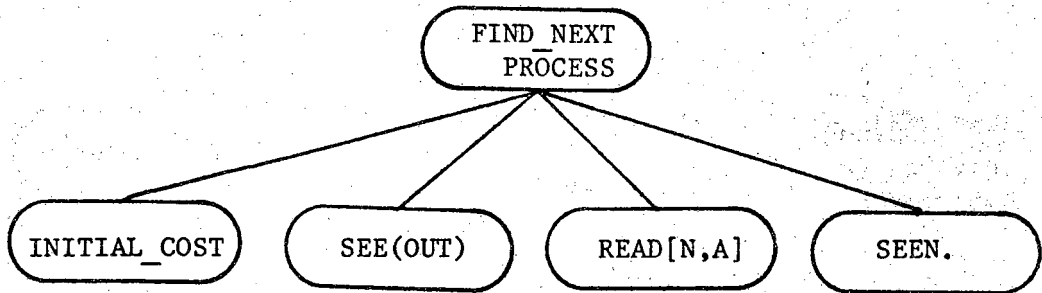
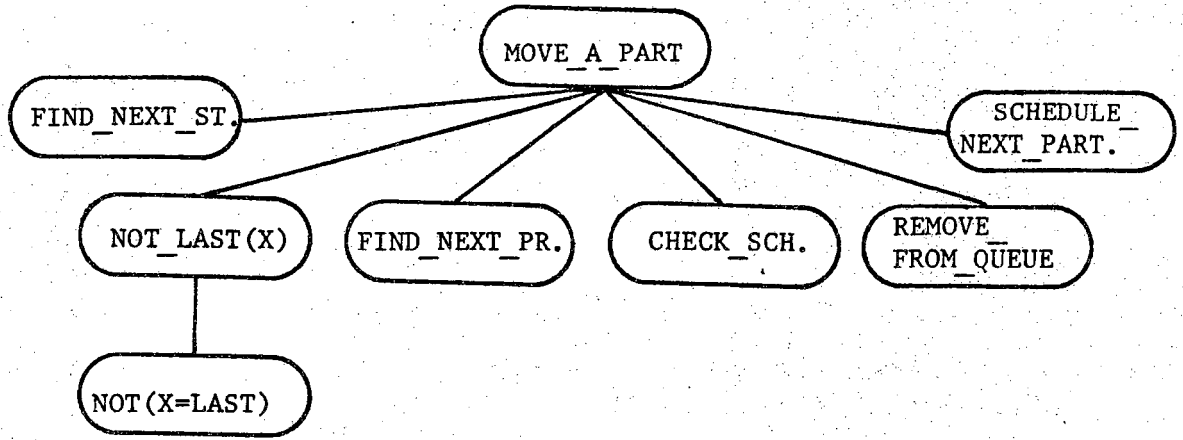
```
edit(s) :- system("vi states").
```

PARTIAL AND/OR TREES OF THE RULES \*



\* All branches are AND branches, and the execution order is from left to right, in a depth first manner.







```
try_schedule_end_of_process(Part, M).

count_queue_size(M, Size) :- queue(M, [List:Tail]),
                             count_tail_size(Tail, I),
                             Size is I + 1.
count_tail_size([L:Tail], I) :-
                             count_tail_size(Tail, R),
                             I is R + 1.
count_tail_size([], 0).
count_queue_size(M, 0).

machine_is_up(M) :- status(M, 1).

update_current_process(Part, M) :- next_process(Part, G),
                                   retract(current_process(Part, _, _)),
                                   asserta(current_process(Part, G, M)).

/* In case of a new part */
create_current_process(Part, M) :- part_pro(Part, [A:T]),
                                   asserta(current_process(Part, A, M)).

/*
*
*
*
*/
/* THIS PART MAKE THE CHANGES WHEN A PROCESS IS FINISHED */
/* First clauses represent dummy parts */
move_a_part(M, Part) :- belong_to(Part, dummyset),
                        part_time(Part, G),
                        current_process(Part, P, M),
                        schedule_dummy(Part, G, P, M),
                        remove_from_queue(M),
                        schedule_next_part(M).

schedule_dummy(Part, G, P, M) :-
    look_for_next_process(Part, G, P, NewP, NewM, NewT),
    not(NewP = last),
    count_queue_size(NewM, Size),
    buffer_size(NewM, G1),
    check_queue(Part, NewP, NewM, NewT, G1, Size),
    retract(current_process(Part, P, M)).
schedule_dummy(Part, G, P, M) :- retract(current_process(Part, P, M)).
/* This was done for process = last */

check_queue(Part, NewP, NewM, NewT, G, Size) :- G > Size,
        ((queue(NewM, [q_time(_, T1):_]), A is T1 + NewT) ;
         (queue(NewM, []), A is NewT)),
        add_to_queue(Part, NewM, A),
        asserta(current_process(Part, NewP, NewM)),
        try_schedule_end_of_process(Part, NewM).
check_queue(Part, NewP, NewM, _, G, Size) :-
    send_to_central_storage(Part),
    asserta(current_process(Part, NewP, NewM)).
look_for_next_process(Part, G, P, NewP, NewM, NewT) :-
    G = [process(P, [time(M, Time)]):Tail],
    (Tail = [process(NewP, [time(NewM, NewT)]):Tail1]
     ; (Tail = [], NewP = last, schedule_new_dummy(Part))), !.
look_for_next_process(Part, G, P, NewP, NewM, NewT) :-
    G = [process(_, [time(M, Time)]):Tail],
    look_for_next_process(Part, Tail, P, NewP, NewM, NewT).
```

```
schedule_new_dummy(Part) :- urand(10, U),
                             introduce_a_part(Part, U).

move_a_part(M, Part) :- find_next_step(Part, X),
                        not_last(X),
                        find_next_process(Part, NewM, A),
                        check_schedule(Part, NewM, A),
                        remove_from_queue(M),
                        schedule_next_part(M), !.

move_a_part(M, Part) :- send_to_assembly(M, Part),
                        schedule_next_part(M).

find_next_step(Part, X) :- next_process(Part, X), !.

not_last(X) :- not(X = last).

find_next_process(Part, NewM, A) :- initial_cost(Part, Z),
                                    see(out),
                                    read([NewM, A]),
                                    seen.

check_schedule(Part, M, A) :- big_number(B),
                              A < B,
                              add_to_queue(Part, M, A),
                              update_current_process(Part, M),
                              try_schedule_end_of_process(Part, M).

check_schedule(Part, M, A) :- update_current_process(Part, M),
                              send_to_central_storage(Part).

send_to_central_storage(Part) :- central_storage(Q),
                                 current_process(Part, CurrentP, _),
                                 append([Part, CurrentP], Q, List),
                                 retract(central_storage(Q)),
                                 asserta(central_storage(List)).

schedule_next_part(M) :- queue(M, A),
                         find_last(A, q_time(Part, _)),
                         current_process(Part, Pro, M),
                         schedule_end_of_process(Part, Pro, M).

schedule_next_part(M). /* Means that the queue is empty */

send_to_assembly(M, Part) :- count_queue_size(assembly, Size),
                             buffer_size(assembly, Q),
                             Size < Q,
                             remove_from_queue(M),
                             retract(current_process(Part, _, M)),
                             expected_time(Part, A),
                             add_to_queue(Part, assembly, A),
                             introduce_a_part(Part, 1),
                             assemble.

send_to_assembly(M, Part).

introduce_a_part(Part, Time) :-
    schedule_event(next_arrival, Part, Time).
```

/\*





```
update_queue_time(T) :- queue(M, A),
                        update_queue_time1(A, T, Nlist),
                        retract(queue(M, A)),
                        asserta(queue(M, Nlist)),
                        fail.
update_queue_time(T).

update_expected_time(T) :- expected_time(X, Y),
                            NewTime is Y - T,
                            retract(expected_time(X, Y)),
                            asserta(expected_time(X, NewTime)),
                            fail.
update_expected_time(T).

update_queue_time1([], T, []).
update_queue_time1(A, T, Nlist) :-
    A = [q_time(Part, Time):Tail],
    Ntime is Time - T,
    update_queue_time1(Tail, T, List),
    append([q_time(Part, Ntime)], List, Nlist), !.

update_event_list :- event_list([A:T]),
                    retract(event_list([A:T])),
                    asserta(event_list(T)).

perform_event(Etime, Ecode, Attrib) :-
    Ecode = next_arrival,
    part_arrival(Attrib), !.
perform_event(Etime, Ecode, Attrib) :-
    Ecode = part_finish,
    (M, Part) = Attrib,
    move_a_part(M, Part), !.
perform_event(Etime, Ecode, Attrib) :-
    Ecode = end_assembly,
    asserta(assembled(Attrib)),
    finished_storage(Q),
    append([(Attrib, Etime)], Q, List),
    retract(finished_storage(Q)),
    asserta(finished_storage(List)),
    retract(robot(busy)),
    asserta(robot(free)),

listing(expected_time),
(not(Attrib = endproduct) ; retract_all_assembled_parts),
!,
assemble.

schedule_end_of_process(Part, Process, Machine) :-
    find_process_time(Part, Process, Machine, ProTime),
    schedule_event(part_finish, (Machine, Part), ProTime).

find_process_time(Part, Process, Machine, ProTime) :-
    part_time(Part, Q),
    find_proper_list(Process, Q, List),
    find_proper_time(Machine, List, ProTime).

find_proper_list(Process, Q, List) :-
    Q = [A:T],
    A = process(Process, List).
find_proper_list(Process, Q, List) :- Q = [A:T],
    find_proper_list(Process, T, List).
```

```
find_proper_time(M,L,ProTime) :- L = [A2!T],
                                A2 = time(M,ProTime).
find_proper_time(M,L,ProTime) :- L = [A2!T],
                                find_proper_time(M,T,ProTime).
```

```
retract_all_assembled_parts :-
                                retract(assembled(wheel)),
                                retract(assembled(axle)),
                                retract(assembled(frontframe)),
                                retract(assembled(frame)),
                                retract(assembled(endproduct)).
```

```
schedule_event(Ecode,Attrib,Time) :-
                                current_time(A),Etime is Time + A,
                                Event = [(Etime,Ecode,Attrib)],
                                event_list(List),
                                append(Event,List,List1),
                                bubble_sort(List1,List2),
                                retract(event_list(List)),
                                asserta(event_list(List2)).
```

```
bubble_sort(L,S):-
                                append(X,[A,B!Y],L),
                                A = (T1,_,_),B = (T2,_,_),
                                (T2 < T1),
                                append(X,[B,A!Y],M),
                                bubble_sort(M,S), !.
```

```
bubble_sort(L,L).
```

```
prepare_output :- tell(out1),current_time(X),write(current_time),tab(5),
                  write(event_list),nl,write(X),tab(5),event_list(Y),
                  write(Y),nl,listing(queue),nl,listing(current_process),
                  nl,write('finished_storage : '),finished_storage(VV),
                  write(VV).
```

```
edit(sim) :- system("vi sim ").
```

---

**Chapter 3**

**Expert Systems in Quality Control**

*Y. S. Chen*

## Expert Systems in Quality Control\*

*Ye-Sho Chen*

### 1. Introduction

#### 1.1. Expert Systems

##### 1.1.1. What Is An Expert System?

According to Weiss and Kulikowski [WEIS 84], an expert system is one that

- (1) handles real-world, complex problems requiring an expert's interpretation
- (2) solves these problems using a computer model of expert human reasoning, reaching the same conclusions that the human expert would reach if faced with a comparable problem.

This artificial intelligence area focuses on developing programs that can operate at human expert levels in restricted problem domains. Such programs use knowledge and inference procedures to solve problems. They can act as consultants backing up human experts, tutors, or program analyzers wading through mountains of data. A typical rule-based expert system program uses sets of production rules, all of which have the following form:

IF <antecedent assertion 1 is true>  
<antecedent assertion 2 is true>

THEN <consequent assertion 1 is true>  
<consequent assertion 2 is true>

---

\*Note that this report will be part of the author's book on Expert System in Quality Control, please don't copy.

Using IF-THEN representation, the quality of the results of an expert system is primarily a function of the size and quality of the knowledge base that it possesses.

Work on expert systems has received extensive attention recently, and prompted growing interest in a wide range of environments. Following is a partial listing of systems, each followed by its function and problem area [GEVA 82]:

ACE repair and maintenance, telephone cable.  
AM concept formation, mathematics;  
CASNET glaucoma diagnosis/therapy, medicine;  
DENDRAL data analysis and interpretation, chemistry;  
EL analysis, electrical circuits;  
GUIDON computer-assisted instruction, medicine.  
HARPY signal interpretation, speech understanding;  
IMS management, automated factory;  
MOLGEN planning, molecular genetics;  
MYCIN diagnosis, medicine;  
NOAH planning, robotics;  
R1 design, computer system configurations;  
SACON user advisor, structural analysis computer program;  
VISIONS image understanding, physics;  
VM monitoring, patient respiration;  
XCEL consultant/intelligent assistant, computer sales;

In addition, a substantial effort is underway to develop expert systems for use in the construction of other expert systems. These include [GEVA 82]:

Programming ROSIE, AGE, HEARSAY III, EMYCIN, OPS 5,  
Tools RAINBOW, KMS, EXPERT, ARBY, MECS-AI, UNITS.

Knowledge TEIRESIAS  
Acquisition EXPERT  
Tools KAS

Learning META-DENTRAL  
Tools EURISKO

### 1.1.2. Why Build An Expert System?

The motivations and advantages for building an expert system are:

- (1) Expert systems are cheap. Human experts are expensive, in short supply,

and when available, can work for limited times and at limited locations.

- (2) With machines that are relatively inexpensive, experts can be "cloned" providing their expertise to many locations simultaneously.
- (3) Expert systems never die. Human experts have limited life span. Expert systems can continue updating the new knowledge if learning ability is available.
- (4) Expert systems can own "universal" expertise. Human experts are limited, in the sense that they can only be expert of some small areas. Integration of diverse expertise can possibly be implemented in an expert system. Some of the earliest medical expert systems, like CASNET and MYCIN, used knowledge from several experts in building their reasoning models.
- (5) Expert systems assist people. Competent individuals in a field can be assisted to operate at expert levels. They can receive expert training by observing the behavior of the expert system and by interacting with the expert system in tutor mode.
- (6) Expert systems help formalize expert knowledge. To build an expert system, we need a formal knowledge representation of the expertise. In formalizing the expert knowledge of how a human expert solves difficult problems with today's best knowledge, we are laying out explicitly how future alternatives can be sought. The expert system thus becomes an empirical tool for experimenting with the representation and uses of knowledge. As such it can make an invaluable contribution to the advancement of practical knowledge.

### 1.1.3. Suitable Tasks for Developing An Expert System

Dr. Steven J. Fenves from Carnegie-Mellon University gave a presentation entitled "Knowledge-Based Expert Systems" at a Purdue University Civil

Engineering seminar on November 13, 1984. In the talk, he pointed out tasks suitable for building an expert system:

- (1) A purely algorithmic solution is not appropriate.
- (2) The task domain must have established experts.
- (3) The experts shall be better than amateurs.
- (4) The task shall not be too easy or too difficult for experts.
- (5) The use of expert systems shall result in considerable savings.

Dr. Fenves gave as an illustration of item (5) the rule of one expert system firm: If it doesn't save \$1,000,000, we won't do it.

Professor Davis of MIT also gave a list of characteristics of good problems where expert systems can be applied [DAVI 84]. The following is part of his list which is different from the previous five.

- (1) The task is primarily cognitive. Medicine and physics qualify; tennis, juggling, and bicycle riding do not.
- (2) The skill is routinely taught to neophytes. It is good if the skill is routinely taught to people who do not know it because it means the experts are accustomed to explaining themselves.
- (3) The task requires no common sense. A good problem involves no common sense. Expert systems based on an established, taught body of knowledge rather than on common sense are more likely to be built successfully.

#### 1.1.4. Process of Building An Expert System

The process of building an expert system is called Knowledge Engineering. The person who builds the system is called Knowledge Engineer. Experience indicates that there are some important differences between the normal

software development cycle as experienced in a typical data processing environment and the process of developing an expert system. The process of knowledge engineering raises new challenges [SAGA 84]. While a complete time-tested methodology for building expert systems does not yet exist, there are some guidelines shared by several knowledge engineers [WEIS 84; SAGA 84].

- (1) Problem identification. Identifying potential applications is perhaps the most subtle and difficult task facing a knowledge engineering group. The eight suitable tasks discussed in the previous section may serve as a good reference.
- (2) Problem assessment. Identifying a suitable problem is usually not enough to justify proceeding. It is necessary to perform an economic analysis of the application as well. Sagalowicz [SAGA 84] gave a technique which he believed will reveal a number of savings. This is to identify the individual, or group of individuals, in the organisation most competent at performing the particular task, then determine the effect of "cloning" that individual to be available round the clock at any location.
- (3) Knowledge acquisition. In this stage, a knowledge engineer is needed to acquire book knowledge or interview experts, abstract the main characteristics of the problem, and then proceed to build a prototype system.
- (4) Choice of knowledge representation. The choice of knowledge representation in computer must have two attributes: power to express the expert knowledge and simplicity to describe, update and explain the knowledge in the model. Several methods are proposed, e.g. predicate logic, semantic network, frame and production system. Among those methods, the IF-THEN production system is widely used.
- (5) Demonstration prototyping. We must recognize that a key success in building an expert system is starting small prototype and building incrementally



to a significantly acceptable system. Demonstration prototypes serve several purposes: first, they allow the knowledge engineer to get enough experience with the application to meaningfully estimate the extent of a project. Second, they provide a test of the suitability of the technical approach taken and gain the feedback from the end-users which may be crucial to later stages of the project. Lastly, the prototype may serve to demonstrate feasibility. A good demonstration may be helpful for securing the resources to build a full-scale operational system.

- (6) Refinement and Performance Evaluation. Once the prototype produces acceptable reasoning, it can be expanded to include more detailed variants of the problems it must interpret. Then it will be tested against more complex cases that will be used as a standard test set for performance evaluation. Many adjustments of the primitive elements and their relationships are bound to come about as the result of this evaluation. The process of refinement and evaluation is iterated until the system is acceptable to cover the entire application.
- (7) Operational Integration. During this phase the system is placed in its actual environment and put to work. Depending on the nature of the application this may involve porting the system to different hardware. Developing and debugging the documentation, training the end-users and creating a maintenance group are part of this phase.

## 1.2. Quality Engineering

- (2) Japan's good product quality is well-known. Americans have compiled an extensive list of explanations to account for the Japanese success including management methods, statistical process control, just-in-time inventory, quality circles and designed experiments using the Taguchi method [COPP 84] and others. The study of all these "keys to success" has been given the

name of Quality Engineering. Dr. Taguchi - Japan's secret weapon [AUTO 84], divides further the quality engineering task into off-line quality control and on-line quality control. In the following subsections, we will describe briefly the two "lines" and propose the potential application areas of expert system techniques.

### 1.2.1. On-line Quality Control and Trouble-Shooting

Traditional quality control programs in the U.S. and elsewhere have relied on a combination of sampling inspections and statistical process control methods. These programs, termed as on-line quality control, have the responsibility to keep processes on the target or nominal value of the specification. Process control deals firstly with the construction of control charts which are used to detect assignable causes failing the quality of the product. When the control charts give the signal of out-of-control, the process is usually shut down and the quality control circle people get together to do trouble-shooting, the second stage of process control. Trouble-shooting is fairly experience-oriented, especially when the process is complicated. The heavy reliance on experience in trouble-shooting makes it amenable to expert systems. Beranek and Newman (BBN) has built an electronic trouble-shooting expert system, call SOPHIE [GEVA 83]. A microtroubleshooter for equipment fault diagnosis which runs on an IBM Personal Computer was developed by Mainwork Ltd [CROA 84;page 10]. In the foreseeable future, we expect more trouble-shooting expert systems to come to the markets.

Two powerful tools are commonly used in doing trouble-shooting. They are: cause-and-effect diagram (or fish-bone diagram) and Pareto diagram. Figure 1 illustrates a cause-and-effect diagram for car repair. The diagram is obtained by translating the rules in [STOC 69]. The mnemonic variables are

listed in Table 1. The complexity of the diagram shows the necessity of experts. Figure 2 demonstrates a Pareto diagram, a cumulation of past experience, of problem in battery systems. So, when we have a battery problem, we may want to check the lead line first. From Figure 1, we see a huge number of production rules, e.g.

```
IF  NSTAR (car won't start)
   and DMOTR (starter doesn't work)
   and DSPKR (speaker doesn't work)
THEN DBATS (problem in battery system)
and
IF  DBATS (problem in battery system)
THEN ILCON (7 10) (7 out of 10, ill-contact in lead line)
     NWBAT (2 10) (2 out of 10, no water in battery)
     DEBAT (1 10) (1 out of 10, dead battery)
```

Using Figure 2, we have ranked priorities when we face uncertainty.

Since the cause-and-effect diagram comes as the result of a "brainstorming process" and the equipment faults are well-defined, there is a high degree of confidence in the reliability of using production rules. Also, the utility of Pareto diagram is feasible, and we don't have the problem of inexact reasoning. In short, trouble-shooting in quality control is a task domain that is well suited for expert system development.

### 1.2.2. Off-line Quality Control and Experimental Designs

The emphasis of on-line quality control has been on tightly controlling manufacturing processes. However, as many products - and their associated manufacturing processes - begin to assume extremely high degrees of complexity, the old "inspect and fix" mode may no longer be suitable. A story illustrating the need of other quality control techniques is given by Mayo [MAYO 84]. One American laboratory was testing a new product design for durability. Components that failed during the tests were simply replaced by better quality (and

more expensive) counterparts. No attempt was made to redesign the product around the less expensive components. The final design released for manufacture therefore exceeded the original budgeted cost.

Today's managers and engineers are forced to redesign their approach to quality control. They have to spend more time on quality engineering activities before sending a product into full-scale production. This approach is called "off-line quality control". The overall goal at this stage is to design a product so that it can be easily and cheaply produced.

According to Taguchi [TAGU 79], off-line quality control advocates a three stage design process; (1) system design, (2) parameter design, and (3) tolerance design. With the development of a new product, the engineers begin with the system design. The system is designed with a specific function. Parameter design attempts to optimize the performance of the system through experimentation. Different parameters in the system are modified for experimentation using the least expensive materials. The final stage of the design process is tolerance design. Only if the product is not acceptable at its optimum level is tolerance design considered.

The Taguchi's design process is fairly knowledge-intensive. Firstly, we need to know the feasible factors and their associated knowledge to compose a good system. Secondly, we need to know the right level of each factor and the corresponding cost structure to achieve the "optimized" states. Finally, at the tolerance design stage, we need to consider higher grade materials or more expensive processes with tighter tolerances. The Taguchi's design process is also experience-oriented. Because in the real world one never gets a right goal immediately, learning is an essential part of the design process.

Knowledge-intensive and experience-oriented tasks reveal the great demand of expert systems in the Taguchi's design process. Recently, Rutgers University held a workshop on knowledge-based design [MOST 84] with the goal of gathering researchers taking an AI approach to automating hardware and software design. Also, AT&T Bell Laboratories is going to host a workshop on AI in statistics with one area of emphasis on the use of expert systems in designed experiments [AT&T 84].

### 1.3. Goal of the Paper

The goal of this paper is to build a prototype equipment fault diagnosis expert system. Also, it is hoped to compare the system performance with other existing ones'. With this goal in mind, the author began to look for a suitable subject. The search procedures are :

- (1) Find out all the quality control books through the computer search terminal of Potter library.
- (2) Browse through all the good journals of quality control.
- (3) Talk to several professors who taught quality control before.
- (4) Read through the two expert system books [HAYE 83 & WEIS 84].

The decision was made after three week's search to focus on "car repair". The reasons for this choice are :

- (1) We know cars better than anything else.
- (2) We have a good manual for car trouble-shooting.
- (3) We can compare our system performance with the system of Weiss and Kulkowski [WEIS 84].

The author also has interest in developing a prototype expert system for off-line

quality control. However, the task is more complicated than the troubleshooting, it takes longer time to develop one. The author intends to have the dream come true in the near future.

## 2. System Organization

Our system composes of four components. The first part will ask question of the user to gather information about the problem. A summary report is printed after the questions are answered. The second part does the interpretive analysis and gives the user treatment recommendation. The "heart" of this step is knowledge base and inference machine which will be examined carefully later. The user may raise why, how and what questions, our system can answer some simple "canned" problems in part three. The final section inquire the confirmation from the user in order to update the uncertainty values in the production rules. Figure 3 gives an overview of our system organization.

### 2.1. Dynamic Questionnaire

In this case questions are posed using a straightforward format, involving multiple choice questions, checklists, or individual numerical or truth-value (yes/no/unknown) responses. The reported results are then summarized. An example is given below.

Example 1.

Hello! You are working on the EXPERT system.

```
*****  
***      BEGINNING OF QUERY      ***  
*****
```

Enter Name or ID Number :Chen

Enter Date of Visit :Dec-10-84

1. Type of Problem:

- (1) Car won't start
- (2) Car starts but problem in engine

(3) Other car problems  
Checklist :(1)

2. Does starter work?  
(1) No  
(2) Yes  
Choose one :(2)

3. Is any fuel in carburetor?  
(1) Yes  
(2) No  
Choose one :(1)

\*\*\*\*\*  
\*\*\* SUMMARY REPORT \*\*\*  
\*\*\*\*\*

Name or ID Number: Chen

Date: Dec-10-84

List of Findings  
-----

(car won 't start)  
(starter works well)  
(fuel in carburetor)

The questionnaire is dynamic, in the sense that we don't ask dummy questions.  
We think that an expert system should possess this characteristic. The other  
example is listed to illustrate the dynamic merit of our questionnaire.

Example 2.

Hello! You are working on the EXPERT system.

\*\*\*\*\*  
\*\*\* BEGINNING OF QUERY \*\*\*  
\*\*\*\*\*

Enter Name or ID Number :Chen

Enter Date of Visit :Dec-4-84

1. Type of Problem:  
(1) Car won't start

- (2) Car starts but problem in engine
  - (3) Other Car Problems
- Checklist :(2)

2. What is the engine problem?
- (1) Engine heat
  - (2) Odorous smell
  - (3) Noise in engine
  - (4) Low speed abnormal
  - (5) High speed abnormal
- Checklist :(30)

3. What kind of noise in engine?
- (1) Pulon
  - (2) Puchi
- Checklist :(1)

\*\*\*\*\*  
 \*\*\*\* SUMMARY REPORT \*\*\*\*  
 \*\*\*\*\*

Name or ID Number: Chen

Date: Dec-4-84

List of Findings

- 
- (engine start but with problem)
  - (noise in engine)
  - (a kind of engine noise sounding like pulon)

2.2. Interpretive Analysis

After the findings are gathered, the system will give diagnostic status and treatment recommendations. The details of the output will be explained later.

Example 1 - Continued.

\*\*\*\*\*  
 \*\*\*\* INTERPRETIVE ANALYSIS \*\*\*\*  
 \*\*\*\*\*

Treatment Priority	Treatment Recommendation
0.3 (30 out of 100)	Adjust the contact of high voltage line
0.2 (20 out of 100)	Replace the spark plug
0.2 (20 out of 100)	Adjust distributor line in contact



- 0.1 ( 10 out of 100) Fix the distributor connector
- 0.1 ( 10 out of 100) Smoothing the platina contact
- 0.06 ( 6 out of 100) Making width of platina contact shorter
- 0.04 ( 4 out of 100) Making width of platina contact larger

Are you interested in the internal symbols?  
Yes/No : Yes

#### Diagnostic Status : Findings to Hypotheses

---

- (rule f-i-3 says variable DIGNS)
- (rule i-h-3 says variable SECON (3 10))
- (rule i-h-3 says variable DSCON (2 10))
- (rule i-h-3 says variable DPLAT (2 10))
- (rule i-h-3 says variable DSPRG (2 10))
- (rule i-h-3 says variable DDIST (1 10))
- (rule i-h-4 says variable PLASM (5 10))
- (rule i-h-4 says variable PLAWL (3 10))
- (rule i-h-4 says variable PLAWS (2 10))

#### Explanation of Variables

---

- (DIGNS : problem in ignition system)
- (SECON : second high voltage lead line is ill-contact)
- (DSCON : lines in distributor are ill-contact)
- (DPLAT : problem in platina contact)
- (DSPRG : problem of spark plug)
- (DDIST : disconnection in distributor)
- (PLASM : platina contact surface is not smooth)
- (PLAWL : width of platina contact is too large)
- (PLAWS : width of platina contact is too small)

#### Treatment Recommendations

---

- (rule h-t-1 says variable FXDIS)
- (rule h-t-2 says variable AJSEL)
- (rule h-t-3 says variable LWPLA)
- (rule h-t-4 says variable SWPLA)
- (rule h-t-26 says variable SMPLA)
- (rule h-t-27 says variable AJDSL)
- (rule h-t-28 says variable RPSPG)

#### Explanation of Variables

---

- (FXDIS : fix the distributor connector)
- (AJSEL : adjust the contact of high voltage line)
- (LWPLA : making width of platina contact larger)
- (SWPLA : making width of platina contact shorter)
- (SMPLA : smoothing the platina contact)

(AJDSL : adjust distributor line in contact)  
(RPSPG : replace the spark plug)

### 2.2.1. Knowledge Base

In our EXPERT system, four representational components are used to design the knowledge base :

- (1) Findings or observations (data),
- (2) Hypotheses or conclusions,
- (3) Treatment Recommendations,
- (4) Reasoning or production rules.

Following the ideas of Weiss and Kulikowski [WEIS 84], there is a sharp distinction between findings and hypotheses. Findings are the observations or measurement results needed to reach conclusion. Hypotheses are the conclusions that may be inferred by the production rules. Treatment recommendations are associated with a measure of priority. The user may try the treatment with the highest priority first. Findings, hypotheses and treatments are reported in the form of mnemonic variables. For example, what follows is a list of variables used in Example 1.

#### Findings

---

(NSTAR : car won't start)  
(MOTDE : starter works well)  
(FCABU : fuel in carburetor)

#### Hypotheses

---

(DIGNS : problem in ignition system)  
(SECON : second high voltage lead line is ill-contact)  
(DSCON : lines in distributor are ill-contact)  
(DPLAT : problem in platina contact)  
(DSPRG : problem of spark plug)  
(DDIST : disconnection in distributor)  
(PLASM : platina contact surface in not smooth)  
(PLAWL : width of platina contact is too large)  
(PLAWS : width of platina contact is too small)

#### Treatments

---

(FXDIS : fix the distributor connector)  
(AJSEL : adjust the contact of high voltage line)  
(LWPLA : making width of platina contact larger)

(SWPLA : making width of platina contact shorter)  
(SMPLA : smoothing the platina contact)  
(AJDSL : adjust distributor line in contact)  
(RPSPG : replace the spark plug)

The reasoning procedures are expressed as production rules, or IF-THEN statements. In terms of our representation, the production rules can be categorized in terms of the three types of logical relationships among findings, hypotheses and treatments :

- (1) FH : finding-to-hypothesis rules,
- (2) HH : hypothesis-to-hypothesis rules,
- (3) HT : hypothesis-to-treatment rules.

In the EXPERT program, FH rules are represented by the LISP variables : f-h-rules and f-i-rules; where f, h and i stand for finding, hypothesis and intermediate hypothesis. The two variables are :

```
(setq f-h-rules
  ((rule f-h-1
    (if ((> variable) SENG)
        ((< variable) NSENG)
        ((< variable) PULON))
     (then ((< variable) DDIST)))

  (rule f-h-2
    (if ((> variable) SENG)
        ((< variable) NSENG)
        ((< variable) PUCHI))
     (then ((< variable) SECON)))

  (rule f-h-3
    (if ((> variable) SENG)
        ((< variable) ABLOW))
     (then ((< variable) PLAWS)))

  (rule f-h-4
    (if ((> variable) SENG)
        ((< variable) ABHIH))
     (then ((< variable) PLAWL)))

  (rule f-h-5
    (if ((> variable) OTHER)
        ((< variable) UNWHE)
        ((< variable) SLONE))
     (then ((< variable) LTIRP (8 10))
           ((< variable) SQBRK (2 10))))

  (rule f-h-6
```

```
(if ((> variable) OTHER)
  ((< variable) UNWHE)
  ((< variable) SLTWO))
(then ((< variable) LWHSW (5 10))
  ((< variable) LJWHP (3 10))
  ((< variable) LGERX (2 10))))
```

(rule f-h-7

```
(if ((> variable) OTHER)
  ((< variable) BRAKE)
  ((< variable) DBRKP))
(then ((< variable) BKFLU (8 10))
  ((< variable) LKBFL (2 10))))
```

(rule f-h-8

```
(if ((> variable) OTHER)
  ((< variable) BRAKE)
  ((< variable) ONBRK))
(then ((< variable) WALYN (6 10))
  ((< variable) GDRMU (4 10))))
```

(rule f-h-9

```
(if ((> variable) OTHER)
  ((< variable) CLUTH)
  ((< variable) NSHFT))
(then ((< variable) LSCLT)))
```

(rule f-h-10

```
(if ((> variable) OTHER)
  ((< variable) CLUTH)
  ((< variable) NHISP))
(then ((< variable) LSCLT)))
```

(rule f-h-11

```
(if ((> variable) OTHER)
  ((< variable) ELECR)
  ((< variable) ODSMK))
(then ((< variable) DSBAT)))
```

(rule f-h-12

```
(if ((> variable) OTHER)
  ((< variable) ELECR)
  ((< variable) SPKNP))
(then ((< variable) SHCON (6 10))
  ((< variable) DRELY (4 10))))
```

(rule f-h-13

```
(if ((> variable) OTHER)
  ((< variable) ELECR)
  ((< variable) FLASH))
(then ((< variable) DFUSE (9 10))
  ((< variable) LSWIR (1 10))))
```

```
(setq f-i-rules
  '((rule f-i-1
    (if ((> variable) NSTAR)
        ((< variable) DMOTR)
        ((< variable) DSPKR))
    (then ((< variable) DBATS))))

  (rule f-i-2
    (if ((> variable) NSTAR)
        ((< variable) DMOTR)
        ((< variable) SPEKR))
    (then ((< variable) DMOTS))))

  (rule f-i-3
    (if ((> variable) NSTAR)
        ((< variable) MOTDE)
        ((< variable) FCABU))
    (then ((< variable) DIGNS))))

  (rule f-i-4
    (if ((> variable) NSTAR)
        ((< variable) MOTDE)
        ((< variable) DCABU))
    (then ((< variable) DFUEL))))

  (rule f-i-5
    (if ((> variable) SENGP)
        ((< variable) HEENG))
    (then ((< variable) DCOOS))))

  (rule f-i-6
    (if ((> variable) SENGP)
        ((< variable) ODORS))
    (then ((< variable) LEKFU))))))
```

For each if-then rule, if there are more than one consequences, then there is a pair of numbers associated with each of them. For example, in rule f-h-5, we have two variables in the then part : LTIRP and SQBRK. The pair (8 10) means 8 out of 10, LTIRP was fired when the antecedents of rule f-h-5 were satisfied. The numbers in the production rules, if any, are obtained from the previous equipment fault records.

HH rules are represented by i-h-rules.

```
(setq i-h-rules
  ((rule i-h-1
    (if ((> variable) DBATS))
    (then ((< variable) ILCON (7 10))
```

```
((< variable) NWBAT (2 10))  
((< variable) DEBAT (1 10))))
```

```
(rule i-h-2  
  (if ((> variable) DMOTS))  
  (then ((< variable) SGEAR (6 10))  
        ((< variable) DIGCL (2 10))  
        ((< variable) BDCON (2 10))))
```

```
(rule i-h-3  
  (if ((> variable) DIGNS))  
  (then ((< variable) SECON (3 10))  
        ((< variable) DSCON (2 10))  
        ((< variable) DPLAT (2 10))  
        ((< variable) DSPRG (2 10))  
        ((< variable) DDIST (1 10))))
```

```
(rule i-h-4  
  (if ((> variable) DPLAT))  
  (then ((< variable) PLASM (5 10))  
        ((< variable) PLAWS (3 10))  
        ((< variable) PLAWL (2 10))))
```

```
(rule i-h-5  
  (if ((> variable) DFUEL))  
  (then ((< variable) VAFUE (4 10))  
        ((< variable) FIFUE (2 10))  
        ((< variable) PIFUE (2 10))  
        ((< variable) PUFUE (1 10))  
        ((< variable) WAFUE (1 10))))
```

```
(rule i-h-6  
  (if ((> variable) DCOOS))  
  (then ((< variable) LMOIL (5 10))  
        ((< variable) BFANB (2 10))  
        ((< variable) BWPIP (2 10))  
        ((< variable) BWTAK (1 10))))
```

```
(rule i-h-7  
  (if ((> variable) LEKFU))  
  (then ((< variable) PIFUE (6 10))  
        ((< variable) PUFUE (4 10))))))
```

Finally, HT rules are represented by h-t-rules.

```
(setq h-t-rules  
  '(  
    (rule h-t-1  
      (if ((> variable) DDIST))  
      (then ((< variable) FXDIS)))  
    (rule h-t-2  
      (if ((> variable) SECON))  
      (then ((< variable) AJSEL)))
```

```
(rule h-t-3
  (if ((> variable) PLAWS))
  (then ((< variable) LWPLA)))
```

```
(rule h-t-4
  (if ((> variable) PLAWL))
  (then ((< variable) SWPLA)))
```

```
(rule h-t-5
  (if ((> variable) LTIRP))
  (then ((< variable) ADPSI)))
```

```
(rule h-t-6
  (if ((> variable) SQBRK))
  (then ((< variable) LSBRK)))
```

```
(rule h-t-7
  (if ((> variable) LWHSW))
  (then ((< variable) FXWSW)))
```

```
(rule h-t-8
  (if ((> variable) LJWHP))
  (then ((< variable) FXWHP)))
```

```
(rule h-t-9
  (if ((> variable) LGERX))
  (then ((< variable) FXGPX)))
```

```
(rule h-t-10
  (if ((> variable) BKFLU))
  (then ((< variable) ABKFL)))
```

```
(rule h-t-11
  (if ((> variable) LKBFL))
  (then ((< variable) ABKFL)))
```

```
(rule h-t-12
  (if ((> variable) WALYN))
  (then ((< variable) DRLYN)))
```

```
(rule h-h-13
  (if ((> variable) GDRMU))
  (then ((< variable) JGDRM)))
```

```
(rule h-t-14
  (if ((> variable) LSCLT))
  (then ((< variable) FXCLT)))
```

```
(rule h-t-15
  (if ((> variable) DSBAT))
  (then ((< variable) CHELC)))
```

```
(rule h-t-16
  (if ((> variable) SHCON))
```

(then ((< variable) RPCON)))

(rule h-t-17  
(if ((> variable) DRELY))  
(then ((< variable) RPRLY)))

(rule h-t-18  
(if ((> variable) DFUSE))  
(then ((< variable) RPRUS)))

(rule h-t-19  
(if ((> variable) LSWIR))  
(then ((< variable) FXWIR)))

(rule h-t-20  
(if ((> variable) ILCON))  
(then ((< variable) FXCON)))

(rule h-t-21  
(if ((> variable) NWBAT))  
(then ((< variable) AWBAT)))

(rule h-t-22  
(if ((> variable) DEBAT))  
(then ((< variable) RPBAT)))

(rule h-t-23  
(if ((> variable) SGEAR))  
(then ((< variable) NPUSH)))

(rule h-t-24  
(if ((> variable) DIGCL))  
(then ((< variable) RPIGL)))

(rule h-t-25  
(if ((> variable) BDCON))  
(then ((< variable) AJBLN)))

(rule h-h-26  
(if ((> variable) PLASM))  
(then ((< variable) SMPLA)))

(rule h-t-27  
(if ((> variable) DSCON))  
(then ((< variable) AJDSL)))

(rule h-t-28  
(if ((> variable) DSPRG))  
(then ((< variable) RPSPG)))

(rule h-t-29  
(if ((> variable) PIFUE))  
(then ((< variable) CLPIF)))



```
(rule h-t-30
  (if ((> variable) WAFUE))
  (then ((< variable) CLFUT)))
```

```
(rule h-t-31
  (if ((> variable) VAFUE))
  (then ((< variable) COCAB)))
```

```
(rule h-t-32
  (if ((> variable) FIFUE))
  (then ((< variable) RPFIL)))
```

```
(rule h-h-33
  (if ((> variable) PUFUE))
  (then ((< variable) RPFUE)))
```

```
(rule h-t-34
  (if ((> variable) LMOIL))
  (then ((< variable) ADMOL)))
```

```
(rule h-t-35
  (if ((> variable) BFANB))
  (then ((< variable) RPBLT)))
```

```
(rule h-t-36
  (if ((> variable) BWPIP))
  (then ((< variable) RPWPI)))
```

```
(rule h-t-37
  (if ((> variable) BWTAK))
  (then ((< variable) RPWTK))))
```

The FH, HH, and HT production rules represent the static knowledge structure of our EXPERT system. The rules are extracted from the cause-and-effect diagram and categorized by content. We will discuss rules-extraction and conflict resolution by content limiting later in Section 3.1.

### 2.2.2. Inference Machine

The control strategy in our EXPERT system is forward-chaining. It starts with a collection of assertions and tries all available rules over and over, adding new assertions as it goes, until no rule applies. Our forward-chaining program is designed by modifying the "stream programming" of Winston and Horn [WINS

84]. Stream programming is a powerful and newly developed concept [ABEL 84 & IDAJ 84]. An example of stream programming in its simplest form is shown in Figure 4. A stream is created when an object is input to a stream generator. The created stream is transformed in several stages, and finally reduced to an object by a reducer. The use of the stream idea is common when talking about input/output operations: programs read information from streams connected to input files and write information to streams connected to output files. It is claimed that a wide class of practical programming can be covered by the stream programming paradigm [IDAJ 84].

Winston and Horn's program was originally designed to handle complicated types of production rules (see their book for examples). When applied to our production rules (much easier type), the idea of the program is the simple "match and fire" strategy. For example, in Example 1, we collected the findings NSTAR, MOTDE and FCABU from the dynamic questionnaire. When the forward-chaining procedure is applied, it loops through the FH rules, i.e. f-h-rules and f-i-rules, first. Since all the antecedents of rule f-i-3 are satisfied, the rule is fired and we get a new assertion DIGNS, the consequence part of rule f-i-3. The procedure then loops through the HH rules, i.e. i-h-rules, and fires rule i-h-3. This time the new assertions are : SECON, DSCON, DPLAT, DSPRG and DDIDT. Since DPLAT satisfies the antecedent part of rule i-h-4, the rule is fired and the variables PLASM, PLAWS and PLAWL are added to the previous assertions. Finally, the procedure loops through HT rules and fire rule h-t-1, h-t-2, h-t-3, h-t-4, h-t-26, h-t-27 and h-t-28. At this moment, the system prints the treatment recommendations, explanation of variables and priority of the treatments.

Note that Winston and Horn's program was written in COMMON LISP with some minor errors in it. We modified it to run on the FRANZ LISP interpreter

stored in the ECN machine.

### 2.3. Questions and Answers

The user may perform some simple questions and answers dialogue in our system. At the end of the interpretive analysis, the system maintains a list of rules successfully used. Each element is to contain the name of the rule, the antecedents, and the conclusion. From this set of rules, the system can answer questions such as (Have you used rule ... ?), (How did you deduce that ... ?), and (Why did you need the assertion ... ?). Besides these, the user may also ask question like (Can you explain ... ?). This procedure explains the mnemonic variables used in the production rules. If the user asks questions besides the four "canned" forms, the system will return: Sorry, I can't answer your question.

Example 1 - Continued.

```
*****  
****   QUESTIONS AND ANSWERS   ****  
*****
```

In this section, you may ask me questions like  
(Have you used rule h-t-1 ?)  
(Can you explain DBATS ?)  
(How did you deduce that AWBAT ?)  
or (Why did you need the assertion RPIGL ?)  
I will answer your questions as soon as possible.  
Thank you!

Question :(Have you used rule i-h-3 ?)  
Answer :Yes! I have used this rule.

Question :(Have you used rule h-t-1 ?)  
Answer :Yes! I have used this rule.

Question :(Have you used rule f-h-5 ?)  
Answer :No! I did not used this rule.

Question :(Can you explain ABCDE ?)  
Answer :(There is no such variable!)

Question :(Can you explain DBATS ?)  
Answer :(DBATS : problem in battery system)

Question :(How did you deduce that AWBAT ?)  
Answer :(AWBAT is not established)

Question :(How did you deduce that NSTAR ?)  
Answer :(NSTAR was given)

Question :(Why did you need the assertion RPIGL ?)  
Answer :(RPIGL was not used)

Question :(Why did you need the assertion NSTAR ?)  
Answer :(NSTAR is needed to show DIGNS)

Question :(Do you love me ?)  
Answer :Sorry, I can't answer your question.

Question :(stop)  
Answer :ok!

#### 2.4. Confirmation and Updating

Since the user may have several choices of treatments with different priorities, the system will ask the user to make confirmation on his decision. The certainty factor in the production rules will be updated.

Example 1 - Continued.

```
*****  
****      CONFIRMATION AND UPDATING      ****  
*****
```

In this section, I need your confirmation on the treatment I gave you above in order to update the uncertainty numbers in the production rules.

Treatment Priority    Treatment Recommendation

0.3 (30 out of 100)	Adjust the contact of high voltage line
0.2 (20 out of 100)	Replace the spark plug
0.2 (20 out of 100)	Adjust distributor line in contact
0.1 (10 out of 100)	Fix the distributor connector
0.1 (10 out of 100)	Smoothing the platina contact
0.06 ( 6 out of 100)	Making width of platina contact shorter
0.04 ( 4 out of 100)	Making width of platina contact larger

Do you accept the 1st treatment?  
Yes/No : No

Do you accept the 2nd treatment?  
Yes/No : Yes

The production rules are updated.

Do you accept the 3rd treatment?

Yes/No : Yes

The production rules are updated.

Do you accept the 4th treatment?

Yes/No : No

Do you accept the 5th treatment?

Yes/No : No

Do you accept the 6th treatment?

Yes/No : No

Do you accept the 7th treatment?

Yes/No : No

Do you want to see the updated production rules?

Yes/No : Yes

The updated production rules are :

```
(setq i-h-rules
  '((rule i-h-1
    (if ((> variable) DBATS))
    (then ((< variable) ILCON (7 10))
           ((< variable) NWBAT (2 10))
           ((< variable) DEBAT (1 10))))
    (rule i-h-2
    (if ((> variable) DMOTS))
    (then ((< variable) SGEAR (6 10))
           ((< variable) DIGCL (2 10))
           ((< variable) BDCON (2 10))))
    (rule i-h-3
    (if ((> variable) DIGNS))
    (then ((< variable) SECON (3 12))
           ((< variable) DSCON (3 12))
           ((< variable) DPLAT (2 12))
           ((< variable) DSPRG (3 12))
           ((< variable) DDIST (1 12))))
    (rule i-h-4
    (if ((> variable) DPLAT))
    (then ((< variable) PLASM (5 10))
           ((< variable) PLAWS (3 10))
           ((< variable) PLAWL (2 10))))
    (rule i-h-5
    (if ((> variable) DFUEL))
    (then ((< variable) VAFUE (4 10))
           ((< variable) FIFUE (2 10))
           ((< variable) PIFUE (2 10))
           ((< variable) PUFUE (1 10))
           ((< variable) WAFUE (1 10))))
    (rule i-h-6
```

```
(if ((> variable) DCOOS))
  (then ((< variable) LMOIL (5 10))
    ((< variable) BFANB (2 10))
    ((< variable) BWPIP (2 10))
    ((< variable) BWTAk (1 10))))
(rule i-h-7
  (if ((> variable) LEKFU))
  (then ((< variable) PIFUE (6 10)) ((< variable) PUFUE (4 10))))
nil
nil))
```

### 3. Knowledge Acquisition

According to Hayes-Roth, Waterman and Lenat [HAYE 83], knowledge acquisition is the transfer and transformation of problem-solving expertise from some knowledge source to a program. The expertise to be elucidated is a collection of specialized facts, procedures, and judgemental rules about the narrow domain area rather than general knowledge about the domain or common-sense knowledge about the world. In general, the knowledge acquisition process is one of the most difficult phases of expert system building.

Dr. George V.E. Otto of AT&T Bell Laboratories in Whippany, New Jersey, gave a talk on "Artificial Intelligence and Expert Systems," in the Chicago ACM Chapter on October 16, 1984. In his seminar, he stated "AI Mantra for the 80's: In the Knowledge Lies the Power; In the Knowledge Acquisition Lies the Bottleneck!!" He then pointed out problems in knowledge acquisition:

- (1) Knowledge engineering is not a well understood process.
- (2) Knowledge engineers are in short supply.
- (3) Knowledge representation language and environment is still evolving.
- (4) Each expert system is still hand crafted.
- (5) Reasons for rules often not coded.
- (6) AI support environments still evolving.

Fortunately, in the equipment fault diagnosis task domain, the knowledge acquisition is relatively easier to deal with. At least three reasons support this :

- (1) The faults of equipment are well-defined, because of their physical nature.
- (2) The use of cause-and-effect diagram captures the expertise in a complete and compact way.
- (3) Pareto diagram solves the uncertainty problem.

In the following, we will discuss how cause-and-effect diagram and Pareto diagram are used in our knowledge acquisition process.

### 3.1. Cause-and-Effect Diagram

To create a cause-and-effect diagram, a team of experts is formed. In a plant, the experts may include design engineers, technicians, line staffs and so on, each of them shall have high degree of understand and long period of experience about the system. After the team is formed, a brainstorming process begins. During the process, various causes are proposed to explain a certain effect. Sometimes previous data or pilot experimentations are necessary to support the explanation. The more complicated the diagram is, the more sophisticated the system is.

In our car repair model, the knowledge was stored in a manual like book. Since it is commercially available and has Chinese version, we assume the book is reliable; in the sense that it was obtained by real experts using an intensive degree of brainstorming process. We then transferred the rules in the manual into a cause-and-effect diagram as shown in Figure 1. We also made a corresponding decision tree because it is commonly used in real world. The tree is shown in Figure 5. Cause-and-effect diagram and decision tree are basically the same. Both diagrams illustrate a classification structure. The only difference may be the former is easier to absorb knowledge of many experts and show clearly the relationship between cause and effect. On the other hand, decision tree is less complicated and easier to be understood by the readers.

From Figure 5, we see how the dynamic questionnaire was designed, how the FH, HH, HT production rules were extracted and how the certainty factors in some rules were obtained.

At the root of the tree, three branches are possible. NSTAR, SENGP and OTHER correspond to car won't start, start but problem in engine and other car problems, respectively. If NSTAR is chosen, then a question with two branches brought to the user. If the user responds with MOTDE, then he receives a question with two possible choices again. At this moment, if he chooses FCABU, then the system stops questioning by listing the findings collected up to this point.

For instance, in our case we have

#### List of Findings

---

#### Explanation of Variables

---

(NSTAR : car won 't start)

(MOTDE : starter works well)

(FCABU : fuel in carburetor)

In the meantime, the system moves to the inference engine by looping through the FH and HH production rules and prints

#### Diagnostic Status : Findings to Hypotheses

---

(rule f-i-3 says variable DIGNS)  
(rule i-h-3 says variable SECON (3 10))  
(rule i-h-3 says variable DSCON (2 10))  
(rule i-h-3 says variable DPLAT (2 10))  
(rule i-h-3 says variable DSPRG (2 10))  
(rule i-h-3 says variable DDIST (1 10))  
(rule i-h-4 says variable PLASM (5 10))  
(rule i-h-4 says variable PLAWL (3 10))  
(rule i-h-4 says variable PLAWS (2 10))

#### Explanation of Variables

---

(DIGNS : problem in ignition system)  
(SECON : second high voltage lead line is ill-contact)  
(DSCON : lines in distributor are ill-contact)  
(DPLAT : problem plaitina contact)



- (DSPRG : problem of spark plug)
- (DDIST : disconnection in distributor)
- (PLASM : platina contact surface in not smooth)
- (PLAWL : width of platina contact is too large)
- (PLAWS : width of platina contact is too small)

Finally, the system moves to the HT rules and gives the treatment recommendations, along with the priority list.

#### Treatment Recommendations

---

- (rule h-t-1 says variable FXDIS)
- (rule h-t-2 says variable AJSEL)
- (rule h-t-3 says variable LWPLA)
- (rule h-t-4 says variable SWPLA)
- (rule h-t-26 says variable SMPLA)
- (rule h-t-27 says variable AJDSL)
- (rule h-t-28 says variable RPSPG)

#### Explanation of Variables

---

- (FXDIS : fix the distributor connector)
- (AJSEL : adjust the contact of high voltage line)
- (LWPLA : making width of platina contact larger)
- (SWPLA : making width of platina contact shorter)
- (SMPLA : smoothing the platina contact)
- (AJDSL : adjust distributor line in contact)
- (RPSPG : replace the spark plug)

#### Priority of the Treatments

---

- 0.3 AJSEL
- 0.2 RPSPG
- 0.2 AJDSL
- 0.1 FXDIS
- 0.1 SMPLA
- 0.06 SWPLA
- 0.04 LWPLA

The use of decision tree (and cause-and-effect diagram), not only shows the rules-extraction, but also demonstrates categorization of rules by content limiting.

### 3.2. Pareto Diagram

We saw from Figure 2, Pareto diagram is a graphical representation device which gives the weight distribution of possible causes. One way to transform a Pareto diagram into a production rule is to assign a pair of data; with the first tuple representing the number of faults for the corresponding variables and the second tuple representing the previous total faults of the antecedents. For example, before updating, the original form of rule i-h-3 is

```
(rule i-h-3
  (if ((> variable) DIGNS))
  (then ((< variable) SECON (3 10))
        ((< variable) DSCON (2 10))
        ((< variable) DPLAT (2 10))
        ((< variable) DSPRG (2 10))
        ((< variable) DDIST (1 10))))
```

Since two variables RPSPG and AJDSL are confirmed (see Example 1 of Section 2.4.), the corresponding hypotheses shall be updated. Because the system kept track of all the rules used, it is easy to trace the rules fired RPSPG and AJDSL.

The two rules are

```
(rule h-t-27
  (if ((> variable) DSCON))
  (then ((< variable) AJDSL)))
```

```
(rule h-t-28
  (if ((> variable) DSPRG))
  (then ((< variable) RPSPG)))
```

Now, the two variables DSCON, DSPRG and the associated consequences of rule i-h-3 are updated. The new rule is

```
(rule i-h-3
  (if ((> variable) DIGNS))
  (then ((< variable) SECON (3 12))
        ((< variable) DSCON (3 12))
        ((< variable) DPLAT (2 12))
        ((< variable) DSPRG (3 12))
        ((< variable) DDIST (1 12))))
```

#### 4. Experimental Results

#### 5. Demonstrative Examples

In the previous sections we decompose the program piecewise to explain the system organization. Now, we demonstrate a complete program to show our system in more details.

Example 3.

```
Script started on Sun Dec 23 21:04:28 1984
$ lisp
Franz Lisp, Opus 38.69
-> (load 'EXPERT)
[load EXPERT]
```

Hello! You are working on the EXPERT system.

```
*****
****      BEGINNING OF QUERY      ****
*****
```

Enter Name or ID Number :Ye-Sho

Enter Date of Visit :Dec-27-84

1. Type of Problem:

- (1) Car won't start
- (2) Car starts but problem in engine
- (3) Other Car Problems

Checklist :(3)

2. Possible other type of problems

- (1) Wheel(s)
- (2) Brake
- (3) Clutch
- (4) Electric

Checklist :(1 '3)

3. How is the wheel unstable?

- (1) Slant in one direction
- (2) Slant right and left

Checklist :(1)

4. What type of brake problem?

- (1) No brake until deep press

(2) Only one side is bad brake  
Checklist :(2)

\*\*\*\*\*  
\*\*\*\* SUMMARY REPORT \*\*\*\*  
\*\*\*\*\*

Name or ID Number: Ye-Sho

Date: Dec-27-84

List of Findings

- (Possible other type of problems)
- (Wheel(s) unstable when car is running)
- (Brake system is in abnormal condition)
- (Wheel slants in one direction)
- (Brake on one side is good the other side malfunctions)

\*\*\*\*\*  
\*\*\*\* INTERPRETIVE ANALYSIS \*\*\*\*  
\*\*\*\*\*

Treatment Priority Treatment Recommendation

- 0.4 (4 out of 10) Add tire pressure
- 0.3 (3 out of 10) Drying the lining of brake
- 0.2 (2 out of 10) Adjust the brake drum gap
- 0.1 (1 out of 10) Loosen the contact of brake

Are you interested in the internal symbols?

Yes/No: Yes

Diagnostic Status : Findings to Hypotheses

- (rule f-h-5 says variable LTIRP (8 10))
- (rule f-h-5 says variable SQBRK (2 10))
- (rule f-h-8 says variable WALYN (6 10))
- (rule f-h-8 says variable GDRMU (4 10))

Explanation of Variables

- (LTIRP : lack of tire pressure)
- (SQBRK : squeeze of brake)
- (WALYN : water in lining of brake system)
- (QDRMU : unbalance of brake drum gap)

Treatment Recommendation

- (rule h-t-5 says variable ADPSI)
- (rule h-t-6 says variable LSBRK)
- (rule h-t-12 says variable DRLYN)

(rule h-t-13 says variable JGDRM)

Explanation of Variables

- (ADPSI : add tire pressure)
- (LSBRK : loosen the contact of brake)
- (DRLYN : drying the lining of brake)
- (JGDRM : adjust the brake drum gap)

\*\*\*\*\*  
 \*\*\* QUESTIONS AND ANSWERS \*\*\*  
 \*\*\*\*\*

In this section, you may ask me questions like  
 (Have you used rule h-t-1?)  
 (Can you explain DBATS?)  
 (How did you deduce that AWBAT?)  
 or (Why did you need the assertion RPIGL?)  
 If you have no more questions, just type "(stop)".

Question : (Why did you need the assertion LTIRP?)  
 Answer : (LTIRP is needed to show ADPSI)

Question : (How did you deduce that ADPSI?)  
 Answer : (ADPSI demonstrated by LTIRP)

Question : (Can you explain ADPSI?)  
 Answer : (ADPSI : add tire pressure)

Question : (You are so stupid?)  
 Answer : Sorry, I can't answer your question.

Question : (Thank you very much!)  
 Answer : Sorry, I can't answer your question.

Question : (stop)  
 Answer : ok!

\*\*\*\*\*  
 \*\*\* CONFIRMATION AND UPDATING \*\*\*  
 \*\*\*\*\*

In this section, I need your confirmation on the treatments I gave you above in order to update the uncertainty numbers in the production rules.

Treatment Priority Treatment Recommendation

- 0.4 (4 out of 10) Add tire pressure
- 0.3 (3 out of 10) Drying the lining of brake
- 0.2 (2 out of 10) Adjust the brake drum gap
- 0.1 (1 out of 10) Loosen the contact of brake

Do you accept the 1st treatment?

Yes/No : Yes

The production rules are updated.

Do you accept the 2nd treatment?

Yes/No : No

Do you accept the 3rd treatment?

Yes/No : No

Do you accept the 4th treatment?

Yes/No : Yes

The production rules are updated.

Do you want to see the updated production rules?

Yes/No : Yes

The updated production rules are :

```
(setq f-h-rules
  '(rule f-h-1
    (if ((> variable) SENGP)
        ((< variable) NSENG)
        ((< variable) PULON))
    (then ((< variable) DDIST)))
  (rule f-h-2
    (if ((> variable) SENGP)
        ((< variable) NSENG)
        ((< variable) PUCHI))
    (then ((< variable) SECON)))
  (rule f-h-3
    (if ((> variable) SENGP) ((< variable) ABLOW))
    (then ((< variable) PLAWS)))
  (rule f-h-4
    (if ((> variable) SENGP) ((< variable) ABHIH))
    (then ((< variable) PLAWL)))
  (rule f-h-5
    (if ((> variable) OTHER)
        ((< variable) UNWHE)
        ((< variable) SLONE))
    (then ((< variable) LTIRP (9 12)) ((< variable) SQBRK (3 12))))
  (rule f-h-6
    (if ((> variable) OTHER)
        ((< variable) UNWHE)
        ((< variable) SLTWO))
    (then ((< variable) LWHSW (5 10))
          ((< variable) LJWHP (3 10))
          ((< variable) LGERX (2 10))))
  (rule f-h-7
    (if ((> variable) OTHER)
        ((< variable) BRAKE)
        ((< variable) DBRKP))
    (then ((< variable) BKFLU (8 10)) ((< variable) LKBFL (2 10))))
```

```
(rule f-h-8
  (if ((> variable) OTHER)
    ((< variable) BRAKE)
    ((< variable) ONBRK))
  (then ((< variable) WALYN (6 10)) ((< variable) GDRMU (4 10))))
(rule f-h-9
  (if ((> variable) OTHER)
    ((< variable) CLUTH)
    ((< variable) NSHFT))
  (then ((< variable) LSCLT)))
(rule f-h-10
  (if ((> variable) OTHER)
    ((< variable) CLUTH)
    ((< variable) NHISP))
  (then ((< variable) LSCLT)))
(rule f-h-11
  (if ((> variable) OTHER)
    ((< variable) ELECR)
    ((< variable) ODSMK))
  (then ((< variable) DSBAT)))
(rule f-h-12
  (if ((> variable) OTHER)
    ((< variable) ELECR)
    ((< variable) SPKNP))
  (then ((< variable) SHCON (6 10)) ((< variable) DRELY (4 10))))
(rule f-h-13
  (if ((> variable) OTHER)
    ((< variable) ELECR)
    ((< variable) FLASH))
  (then ((< variable) DFUSE (9 10)) ((< variable) LSWIR (1 10))))
nil
nil))
```

```
t
-> ^D
Goodbye
$ ^D
script done on Sun Dec 23 21:08:43 1984
```

### 5.1. Performance Evaluation

Performance evaluation is an important area of building expert systems. Weiss and Kulikowski [WEIS 84], and Hayes-Roth and others [HAYE 83] devoted several spaces discussing this topic in their building expert systems books. Professor King-Sun Fu of Purdue University believed at least three topics should be covered in discussing performance evaluation. They are : reasoning power, cost

and learning.

As stated in Section 1.3., we select car trouble-shooting as a topic to compare the car repair model of Weiss and Kulikowski [WEIS 84]. Although their car repair model is not complete, we may compare the two systems from the design point of view.

#### 5.1.1. Reasoning Power

There are similar characteristics on reasoning power between the two systems :

- (1) The production rules are composed of FH, HH and HT. The advantage of this device is to resolve the rules conflict by content limiting. Also, the partition of the production rules into three categories, saves the execution time of the inference engine. Otherwise, the system will loop through all the production rules, some of them might be irrelevant.
- (2) Forward-chaining is used as the control strategy.
- (3) Both systems have simple explanation ability. Matching the query with certain template is used as question understanding device. Generation of canned text is used to answer question.

There are differences on reasoning power between the two systems

- (1) Knowledge acquisition. We use a cause-and-effect diagram and a Pareto diagram to collect expertise. Weiss did not specify exactly how their production rules were obtained. In this sense, our system is easier to be understood and updated. These shall be important factors when we talk about reasoning power.



- (2) Collection of findings. We use dynamic questionnaire to gather findings. A good feature of this approach is to avoid asking irrelevant questions. Quite differently, Weiss used a fixed format of questionnaire. We ask questions as most human experts do; in terms of cause-and-effect diagram, we start from the effect part first and trace to the possible causes leading to the effect. We continue the effect-first-cause-next process until we hit the leaf of the diagram. Conversely, they start from the leaves and move to the body of the diagram. Figure 6 and Table 2 show the cause-and-effect diagram of their car repair model. From this figure, we see their questionnaire was designed by partitioning all the leaves of the diagram into several groups. Each group represents a question shown on page 78 of their book. One drawback of their approach is that questionnaire shall be changed, when new knowledge is added.
- (3) Certainty factor. In our system, the use of uncertainty numbers is objective and more reliable. This is because we use the idea of Pareto diagram which enables the updating ability of the production rules. Conversely, their certainty factor is subjective and unreliable. As we all know certainty factor is a very important ingredient of reasoning power.

### 5.1.2. Cost

Since our system is written in LISP, the cost of running our system shall be cheaper than Weiss's FORTRAN-oriented system. However, execution speed is not the only factor in judging the use of programming languages. Several other factors shall also be considered. One reason that FORTRAN was used is because this language has more users than the programming languages for AI, e.g. LISP and PROLOG. According to the advertisements in the first issue of International Journal of Expert Systems and Knowledge Engineering [CROA 84], several dedicated FORTRAN machines are commercially available for building expert systems.

The author believes to talk about the cost of expert systems, one should consider what kinds of functions the expert system can perform. This is similar to buying a personal computer in today's market.

### **5.1.3. Learning**

Both of the systems don't have the ability of learning. The ability to learn is still a big research area in Artificial Intelligence.

## **6. Discussions and Conclusions**

### **6.1. Discussions**

In the following we will discuss several interesting topics related to this paper. These include: beyond the prototype model, human fault diagnostic models, and comparison of expert system program and conventional one.

#### **6.1.1. Beyond the Prototype Model**

As we mentioned in section 1.1.4., further testings, refinements and generalizations are needed to shape a prototype expert system. In Figure 7, Dr. Davis [DAVI 84] showed six states of development for some well-known expert systems. Ranging from "a gleam in somebody's eye" to wide commercial use, five systems have reached the stage of commercial use. But a larger number of programs are somewhere between the stage of debugged program and experimental use. Davis continued to make a note on R1 which has by far the most clearly defined development process, evolving through a sequence of stages similar to those listed in below.

- (1) System design
- (2) System development
- (3) Formal evaluation of performance

- (4) Formal evaluation of acceptance
- (5) Extended use in prototype environment
- (6) Development of maintenance plans
- (7) System release

In its first formal evaluation the system was tested on approximately twenty cases. The results suggested that R1 would soon solve problems correctly 90 percent of the time. But when it was distributed to its user community for more extended testing, users criticized the system performance 40 percent of the time. What happened? Some of those problems were mistakes on the part of the users resulting from incorrect data or a misunderstanding of program operation. But there was a more basic lesson: research environments, no matter how carefully tailored, are not identical to user environments. John McDermott, one of the developers of R1, has said that R1's knowledge base grew at least as much during the final stage of its development as it did in any of the previous stages of its development.

The same theme shall apply to our trouble-shooting expert system. One drawback of the system is the expertise came from a book knowledge. We need criticisms from real car repair experts and further improvements if the system is to be accepted in the market place.

#### **6.1.2. Human Fault Diagnostic Models and Second Generation Expert Systems**

Rouse and his colleagues [ROUS 83 & ROUS 84] made a series of investigations of human problem solving performance in fault diagnosis tasks. They attempted to synthesize a model capable of describing human problem solving in general. This model, based on a thorough review of the problem solving literature, is conceptually proposed recently. The model operates on three levels:

- (1) Recognition and classification. This is the process whereby humans determine the problem solving situations with which they are involved. Familiar situations may invoke a standard "frame" [MINS 75] while unfamiliar situations may lead to the use of analogies or even basic principles of investigation.
- (2) Planning. This may involve the use of familiar "scripts" [SCHA 79] or, if no script is available, require generation of alternatives, imagining of consequences, valuing of consequences, etc [JOHA 79].
- (3) Execution and monitoring. This involves the use of so called symptomatic rules (S-rules) and topographic rules (T-rules). This dichotomy between symptomatic and topographic problem solving was formalized in a fuzzy rule-based model. This model first attempts to find familiar patterns among the symptoms of the failure (i.e., among the state variables of the system). If a match is found, S-rules are used to map directly from symptoms to hypothesized failure. If there are no familiar patterns among the state variables, the model uses T-rules to search the structure (i.e., functional relationships) of the system.

What Rouse and his colleagues did matches the research direction of expert systems, the so called second-generation. Second-generation systems are those that perform the same task already accomplished by another system, but do so using some variation in representation of the knowledge or control structure, or both. One goal of second-generation systems is experimentation: to demonstrate, by doing the same task done by the first-generation system, whether the altered knowledge representation or control structure results in improved performance. A good example of second-generation vs first-generation expert system is CENTAUR vs PUFF [AIKI 83]. PUFF was written to perform pulmonary function test interpretations. The structure used to represent knowledge in

PUFF were IF-THEN production rules. CENTAUR is better than PUFF because the former represents its knowledge as a combination of frames and production rules and performs tasks more flexible and powerful than the latter.

Evidently, our trouble-shooting expert system belongs to the first-generation. To upgrade our system in an expert sense, one good way may be to implement Rouse and his associates' conceptual model. This will be progressed by the author in the near future.

### 6.1.3. Comparison of Expert System Program and Traditional One

One trivial question a beginner might ask in learning expert systems is "What is the difference between an expert system from an ordinary computer application?". A short answer to this question is that ordinary computer programs organize knowledge on two levels: data and program. Most expert systems organize knowledge on three levels: data, knowledge base, and control strategy.

More clearly, what distinguishes an expert system from an ordinary computer application is that in a conventional computer program, pertinent knowledge and the methods for utilizing it are all intermixed. In an expert system, the problem-solving model appears explicitly as a knowledge base rather than implicitly as part of the coding, and the knowledge base is manipulated by a separate, clearly identifiable control strategy. For example, in our car repair expert system, we have global data base, knowledge base and inference machine clearly separated (See Figure 3).

One important reason for the modularization is that each part may need to be updated, edited and deleted frequently. The global data base is the working memory of the system. It keeps track of input data for the current problem, the status of the current problem, and the relevant history of what has been done so far. The knowledge base stores the required knowledge of the tasks. Extendibil-

ity, simplicity, and explicitness are basic requirements to consider when choosing a knowledge representation scheme. The inference machine is actually a collection of procedures which makes decisions about how to use the system's knowledge by organizing and controlling the steps. The procedures are modified whenever a more powerful algorithm is released. For example, a new algorithm, RETE developed by Forgy at Carnegie-Mellon University [FORG 81], is claimed recently to have a nice feature. Large production systems that use the RETE algorithm generally execute much faster than similar systems without it.

Someday, we might modify our inference engine by using RETE algorithm.

## 6.2. Conclusions

Quality control is a well-suited and cost-effective area for the development of expert systems. In this report, the author showed a prototype expert system for trouble-shooting in on-line quality control. Also, a prototype system for experimental design in off-line quality control is in preparation. Besides continuing shaping the prototype model by testing and refinement, we need to generalize the model by closely keeping track of the newly developed technology. Listed below are part of them:

- (1) Knowledge representation. Since real experts use more than rules, the use of production rules only represents surface-structure knowledge of human experts. Some causal models representing deep-structure knowledge are being pursued, under the title of Cognitive Science, by some leading AI scientists.
- (2) Inference machine. Besides forward-chaining, there are several other inference procedures commonly used, e.g. backward-chaining, pattern matching, generate-and-test, and constraint satisfaction. The use of these various procedures depends on the nature of the problem tasks. It would be

interesting to see a paper discussing the use of each inference procedure in its best working environment.

- (3) Learning. The field of machine learning strives to develop methods and techniques to automate the acquisition of new information, new skills, and new ways of organizing existing information. As Simon [SIMO 83] has pointed out, if learning could be automated and the results of that learning transferred directly to other machines which could further augment and refine the knowledge, one could accumulate expertise and wisdom in a way not possible by humans - each individual person must learn all relevant knowledge without benefit of a direct copying process.
- (4) Programming tools. Brown [BROW 84], a pioneer researcher in AI, says that the success of AI and expert systems rests not in the intellectual arena but in the recent dramatic advances in hardware, particularly hardware that can effectively execute LISP - the lingua franca of AI. He continued, "AI systems that required dedicated, million-dollar mainframes five years ago now can run on machines that cost only \$25,000. For the first time we have cost-effective delivery engines for expert systems, a major change." Besides LISP, several programming languages are suited for expert systems development; e.g., PROLOG, and OPS 5. The RETE algorithm mentioned above is used in the OPS family of production system languages.
- (5) Natural language front-end. Expert systems that need to produce English explanations have for the most part gotten along quite well so far on simple, small, linguistically naive generation programs. Full-scale, linguistically motivated text generation capabilities will be required of expert systems simply because the intricate grammatical constraints on the text produced by recursively embedded source data structures will eventually swamp any ad hoc facility, especially as the number of objects in the system requiring

description grows and the sophistication of the desired text increases. This trend will lead to a better explanation and consultation ability for expert systems.

In summary, the feasibility of using expert systems technology in quality control is without questions. However, to build a real expert system, it is necessary to do intensive testing, constant refinement, and incremental generalization as a continuing task during the entire useful life of the system.



## 7. References and Appendix

### 7.1. References

[ABEL84]

Abelson, H. and Gerald J.S. "Structure and Interpretation of Computer Programs," MIT Press, Cambridge, Massachusetts, 1984.

[AIKI]

Aikins, J.S. "Prototypical Knowledge for Expert Systems," Artificial Intelligence, pp.163-210, 1983.

[AT&T84]

"Workshop on Artificial Intelligence in Statistics," AT&T Conference Center, Princeton NJ, 15-16 April 1985. (See the Ad in AI Magazine, p.87, Fall 1984)

[AUTO84]

"Dr. Taguchi - Japan's Secret Weapon," August 1984 Issue of Automotive Industries, pages 18 and 20.

[BROW84]

Brown, J.S. "The Low Road, the Middle Road, and the High Road," In the AI Business, Commercial Uses of AI, 1984.

[BUCH84]

Buchanan, B.G. and Shortliffe, E.H., "Rule-Based Expert Systems : The MYCIN Experiments of the STANFORD Heuristic programming project," Reading, MA : Addison-Wesley, 1984.

[CORP]

Copp Richard, "Report on Japanese Quality Engineering using Designed Experiments," July 1984, American Supplier Institute, Incorporated.

[CROA84]

Croall, I., Ishizuka, M., and Watterman, D. (Eds.) "EXPERT SYSTEMS", Vol.1, No.1, July 1984.

[DAVI84]

Davis Randall, "Amplifying Expertise with Expert Systems," The AI Business, Commercial Uses of Artificial Intelligence, edited by P.H. Winston and K.A. Prendergast, 1984.

[FORG81]

Forgy, C.L. "OPS5 user's manual," Technical Report, Carnegie-Mellon University, Department of Computer Science, 1981.

[GEVA82]

Gevarter, W.B. "An Overview of Expert Systems," (NBSIR 82-2505) Washington, DC: National Bureau of Standards; 1982.

[GEVA83]

Gevarter, W.B. "Expert Systems : Limited but Powerful," Computers, 1983.

[HAYE83]

Hayes-Roth, F., Waterman, D., and Lenat, D. (editors) "Building Expert Systems," Addison-Wesley, New York, 1983.

[IDAJ84]

Ida, J. and Tanaka, J. "Functional Programming with Streams," New Generation Computing, 2 (1984) 261-275, OHMSHA, LTD. and Spring-Verlag.

[JOHA79]

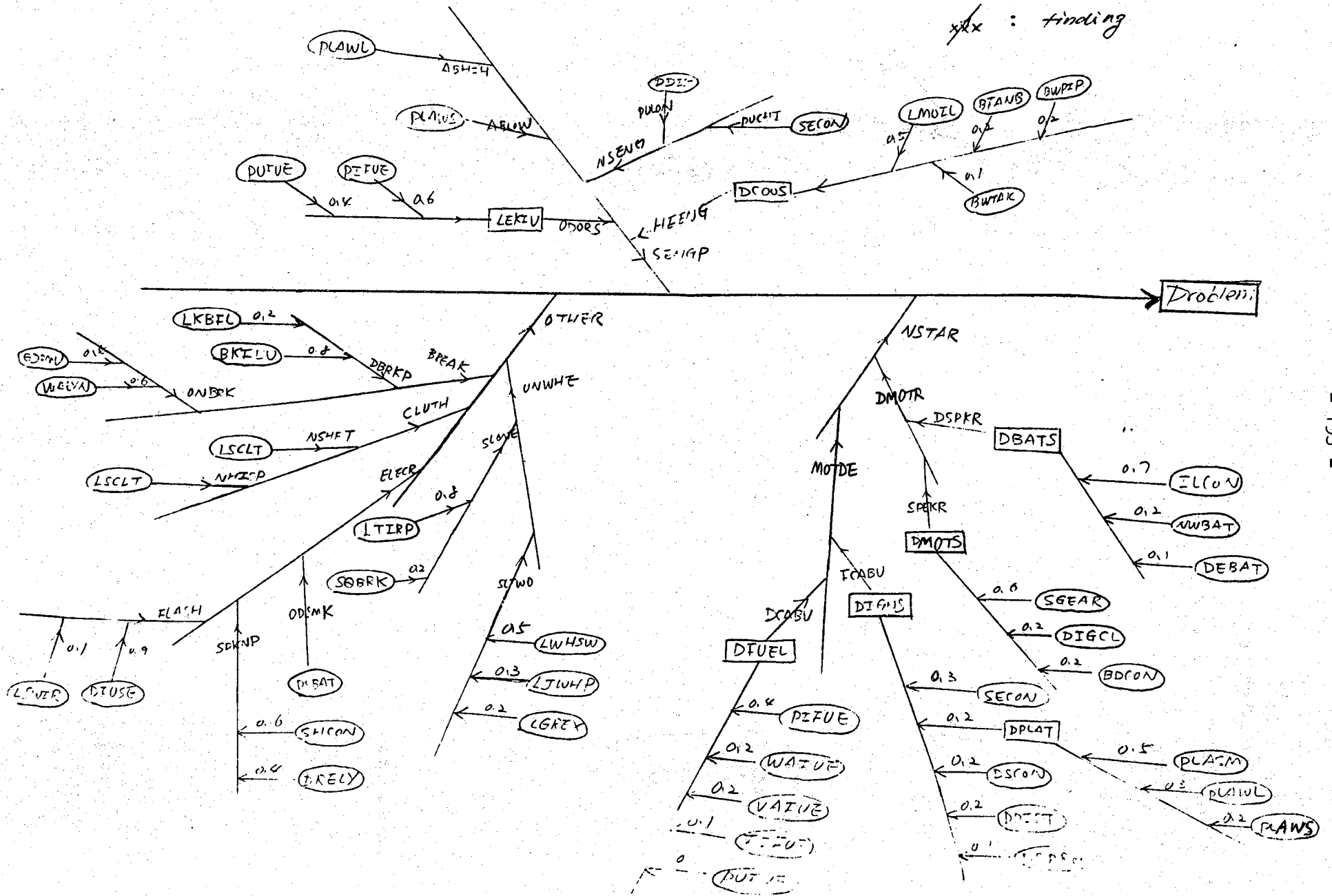
Hohannsen, G. and Rouse, W.B. "Mathematical Concepts for Modeling Human Behavior in Complex Man-Machine Systems," Human Factors, 21(6):733-747,

- 1979.
- [MAYO84]  
Mayo John, "Process Design as Important as Product Design," Manger's Journal, The Wall Street Journal, 1984.
- [MINS75]  
Minsky, M. "A Framework for representing knowledge," In P.H. Winston (Ed.), The Psychology of Computer Vision. New York: McGraw-Hill, 1975.
- [MOST84]  
MOSTOW Jack, "Rutgers Workshop on Knowledge-Based Design," ACM SIGART newsletter, pp. 19-32, #90, October 1984.
- [ROUS]  
Rouse, W. B., "Models of Human Problem Solving: Detection, Diagnosis, and Compensation for System Failures", Automatica, vol. 19, 1983, pp. 613-625.
- [ROUS84]  
Rouse, W. B. and Ruston, M. H., "Human Problem Solving in Fault Diagnosis Tasks", Advances in Man-Machine Systems Research, Vol. 1, 1984.
- [SAGA84]  
Sagalowicz, D. "Development of an Expert System," EXPERT SYSTEMS: The International Journal of Knowledge Engineering; Vol.1, No.2, October 1984, pp.137-141.
- [SCHA77]  
Schank, R.C. and Abelson, R.P. "Scripts, Plans, Goals, and Understanding," Hillsdale, NJ: Lawrence Erlbaum, 1977.
- [SIMO83]  
Simon, H.A. "Why Should Machines Learn?" In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.). Machine Learning: An Artificial Intelligence Approach. Palo Alto, CA: Tioga Press; 1983; 25-37.
- [STOC69]  
Stockel, M. W., "Auto Service and Repair", Good-Heart Willcox Company, Inc., S. Holland, IL.
- [TAGU79]  
Taguchi, Genichi. "Introduction to Off-Line Quality Control," June 1979, American Supplier Institute, Incorporated.
- [WEIS84]  
Weiss, S.M. and Kulikowski, C.A. "A Practical Guide to Designing Expert Systems," Rowman & Allanheld, Publishers, 1984.
- [WINS84]  
Winston, P.H. and Horn, B.K.P. "LISP", 2nd edition, Addison-Wesley Publishing Company, 1984.

8.1 Figures and Tables

□ : intermediate hypothesis

xxx : finding



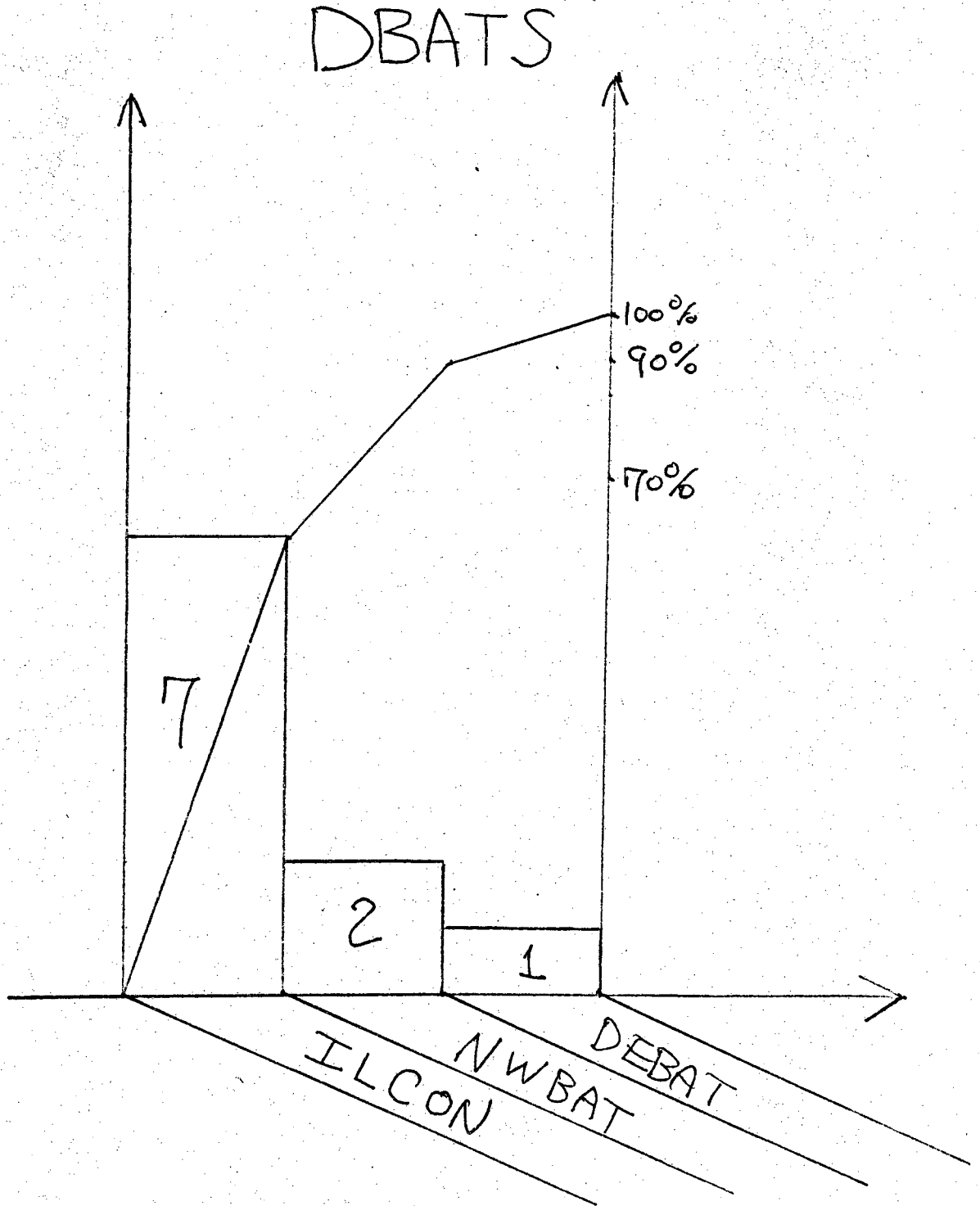


Figure 2. Pareto Diagram of Battery Defects.

Table 1. Explanation of the Abbreviated Variables in our Car Repair Model.

- (ABHIH : engine abnormal in high speed but normal in low speed)
- (ABKFL : add brake fluid)
- (ABLOW : engine abnormal in low speed but normal in high speed)
- (ADMOL : add motor oil)
- (ADPSI : add tire pressure)
- (AJBLN : loosen the lead line clean and fix it)
- (AJDSL : adjust distributor line in contact)
- (AJSEL : adjust the contact of high voltage line)
- (AWBAT : adding water in battery)
- (BDCON : battery lead line ill-contact)
- (BFANB : loose or broken of fan belt)
- (BKFLU : lack of brake fluid)
- (BRAKE : brake system in abnormal condition)
- (BWPIP : water pipe broken)
- (BWTAK : water tank broken)
- (CHELC : check electric circuit if short circuit exist)
- (CLFUT : clear fuel tank)
- (CLPIF : clean the fuel pipe line)
- (CLUTH : problem in clutch)
- (COCAB : cool carburetor)
- (DBATS : problem in battery system)
- (DBRKP : no brake until deep press on accelerator)
- (DCABU : no fuel in carburetor)
- (DCOOS : problem in coolant system)
- (DDIST : disconnection in distributor)
- (DEBAT : dead battery)
- (DFUEL : problem in fuel system)
- (DFUSE : fuse is dead)
- (DIGCL : ignition coil does not work)
- (DIGNS : problem in ignition system)
- (DMOTR : starter does not work)
- (DMOTS : problem in motor system)
- (DPLAT : problem in platinum contact)
- (DRELY : relay is out of order)
- (DRLYN : drying the lining of brake)
- (DSBAT : disconnecting battery)
- (DSCON : lines in distributor are ill-contact)
- (DSPKR : speaker does not work)
- (DSPRG : problem of spark plug)
- (ELECR : problem in electric system)
- (FCABU : fuel in carburetor)
- (FIFUE : block of fuel filter)
- (FLASH : flash light is out of order)
- (FXCLT : fix the clutch)
- (FXCON : fix the connector)
- (FXDIS : fix the distributor connector)
- (FXGPX : fix the gear box screw)
- (FXWHP : fix the joint between wheel and propeller)
- (FXWIR : fix lead wire to light)
- (FXWSW : fix wheel screw)
- (GDRMU : unbalance of brake drum gap)

(HEENG : car start but engine has problems)  
(ILCON : ill-contact in lead line)  
(JGDRM : adjust the brake drum gap)  
(LEKFU : fuel leakage)  
(LGERX : loosen of gear box screw)  
(LJWHP : loosen between wheel and propeller shaft)  
(LKBFL : leakage of brake fluid)  
(LMOIL : lack of motor oil)  
(LSBRK : loosen the contact of brake)  
(LSCLT : loosen of clutch)  
(LSWIR : loosen of wire)  
(LTIRP : lack of tire pressure)  
(LWHSW : loosen of wheel screw)  
(LWPLA : making width of platina contact larger)  
(MOTDE : starter works well but engine doesn't work)  
(NHISP : no high speed)  
(NPUSH : put transmission in N and push the car)  
(NENG : noise in engine)  
(NSHFT : noise in shifter)  
(NSTAR : car won't start)  
(NWBAT : no water in battery)  
(ODORS : odorous smell in engine)  
(ODSMK : odorous smell and smoke when car running)  
(ONBRK : one side brake is good the other side is malfunction)  
(OTHER : possible other type of problems)  
(PIFUE : block of fuel pipe)  
(PLASM : platina contact surface is not smooth)  
(PLAWL : width of platina contact is too large)  
(PLAWS : width of platina contact is too small)  
(PUCHI : a kind of engine noise sounding like puchi)  
(PULON : a kind of engine noise sounding like pulon)  
(PUFUE : fuel pump is dead)  
(RPBAT : replace a battery)  
(RPBLT : replace the fan belt)  
(RPCON : replace the connector)  
(RPFIL : replace fuel filter)  
(RPFUE : replace the fuel pump)  
(RPFUS : replace fuse)  
(RPIGL : replace the ignition coil)  
(RPRLY : replace the relay)  
(RPSPG : replace the spark plug)  
(RPWPI : replace water pipe line)  
(RPWTK : replace water tank)  
(SECON : second high voltage lead line is ill-contact)  
(SENGP : engine start but with problem)  
(SGEAR : motor gear and fly gear squeeze)  
(SHCON : short circuit of connector)  
(SLONE : wheel slant in one direction)  
(SLTWO : wheel slant right and left two direction)  
(SMPLA : smoothing the platina contact)  
(SPEKR : speaker sound well)  
(SPKNP : speaker sound and can not be stopped)  
(SQBRK : squeeze of brake)  
(SWPLA : making width of platina contact shorter)

**(UNWHE : wheel unstable when car is running)**  
**(VAFUE : verporization of fuel in carburetor)**  
**(WAFUE : water in fuel tank)**  
**(WALYN : water in lining of brake system)**

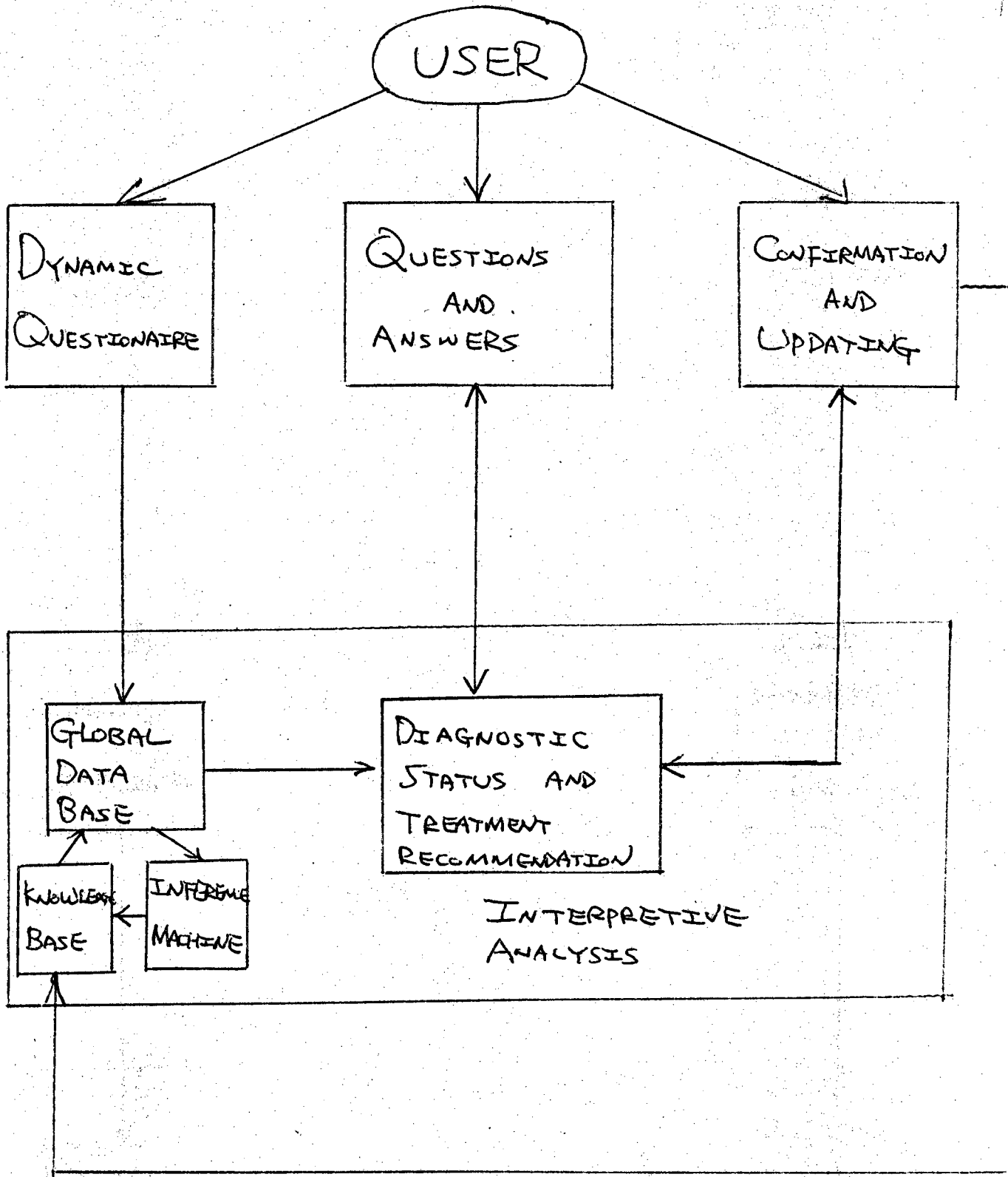
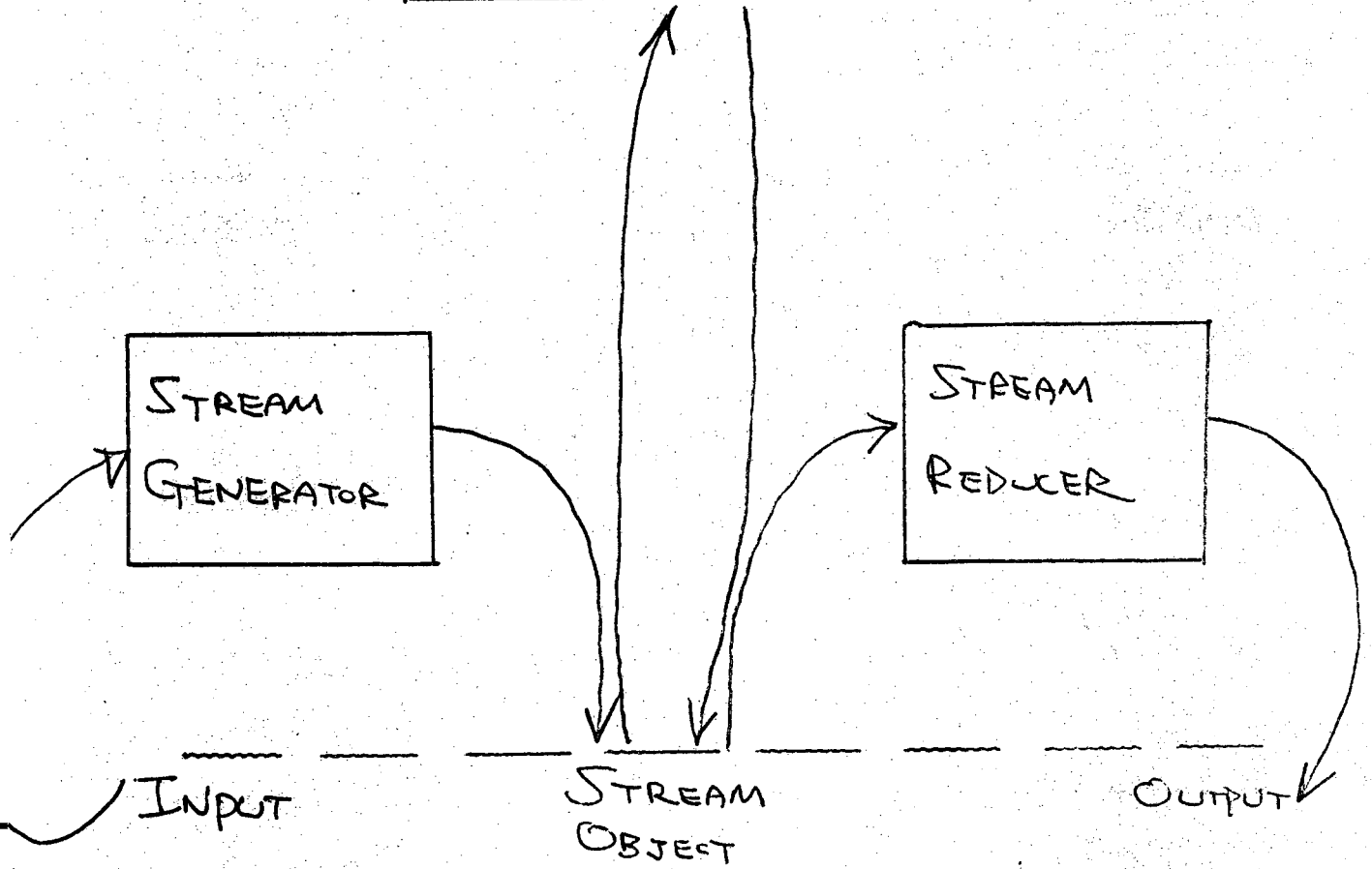
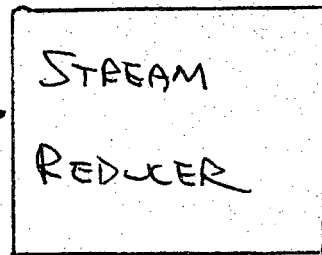
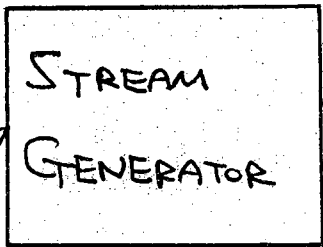
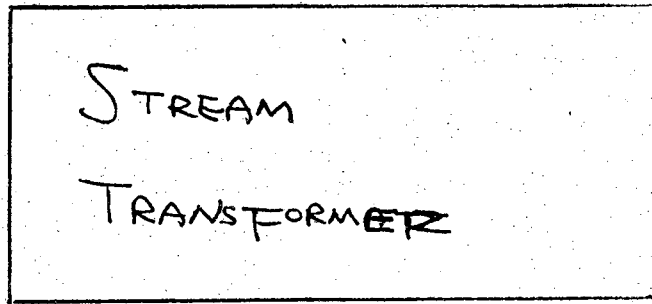


Figure 3. System Organization of Trouble-Shooting Expert System.



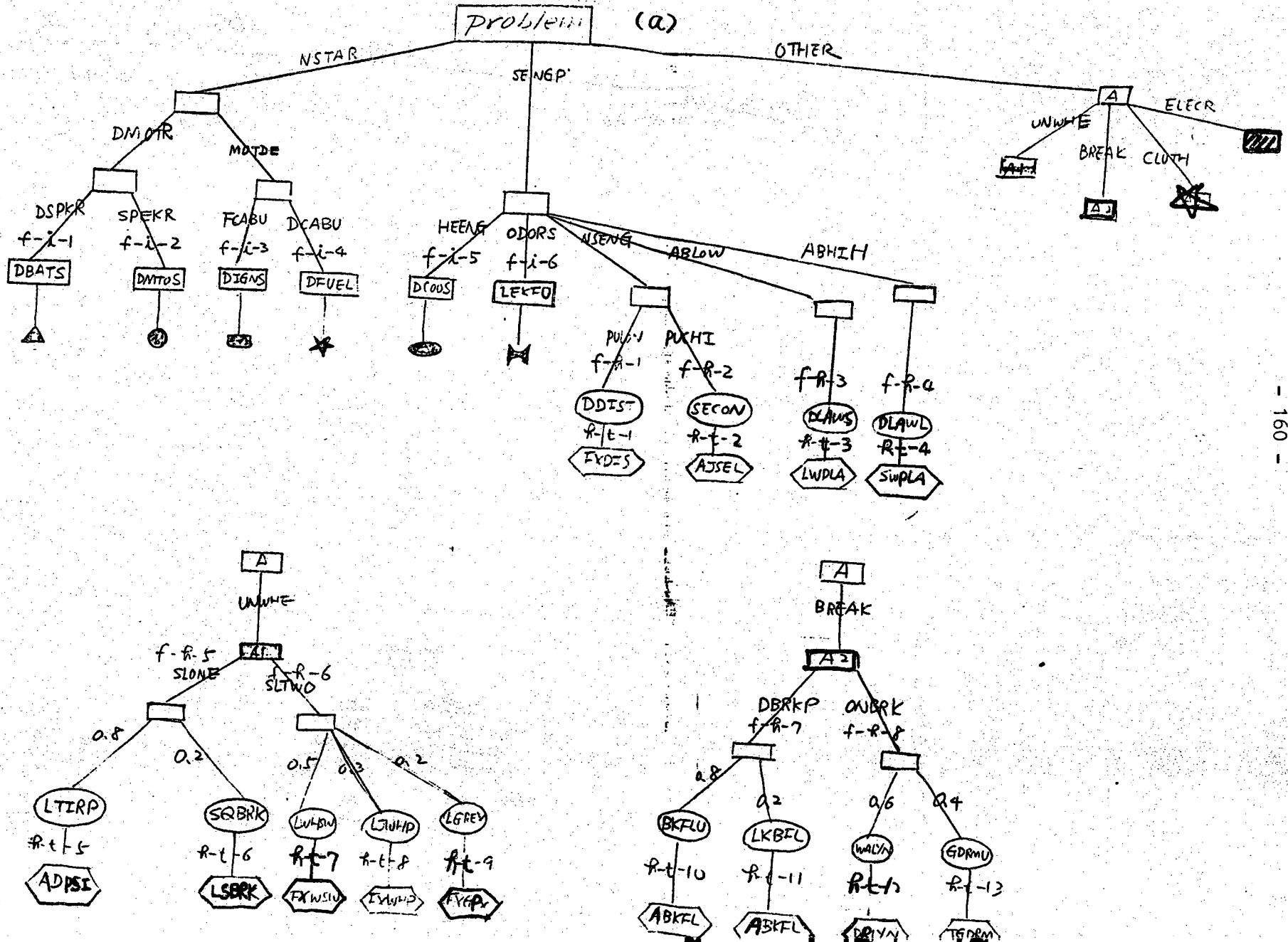
# FUNCTION WORLD



# OBJECT WORLD

Figure 4. Stream Programming.

Figure 5. Decision Tree of the Car Trouble-Shooting.



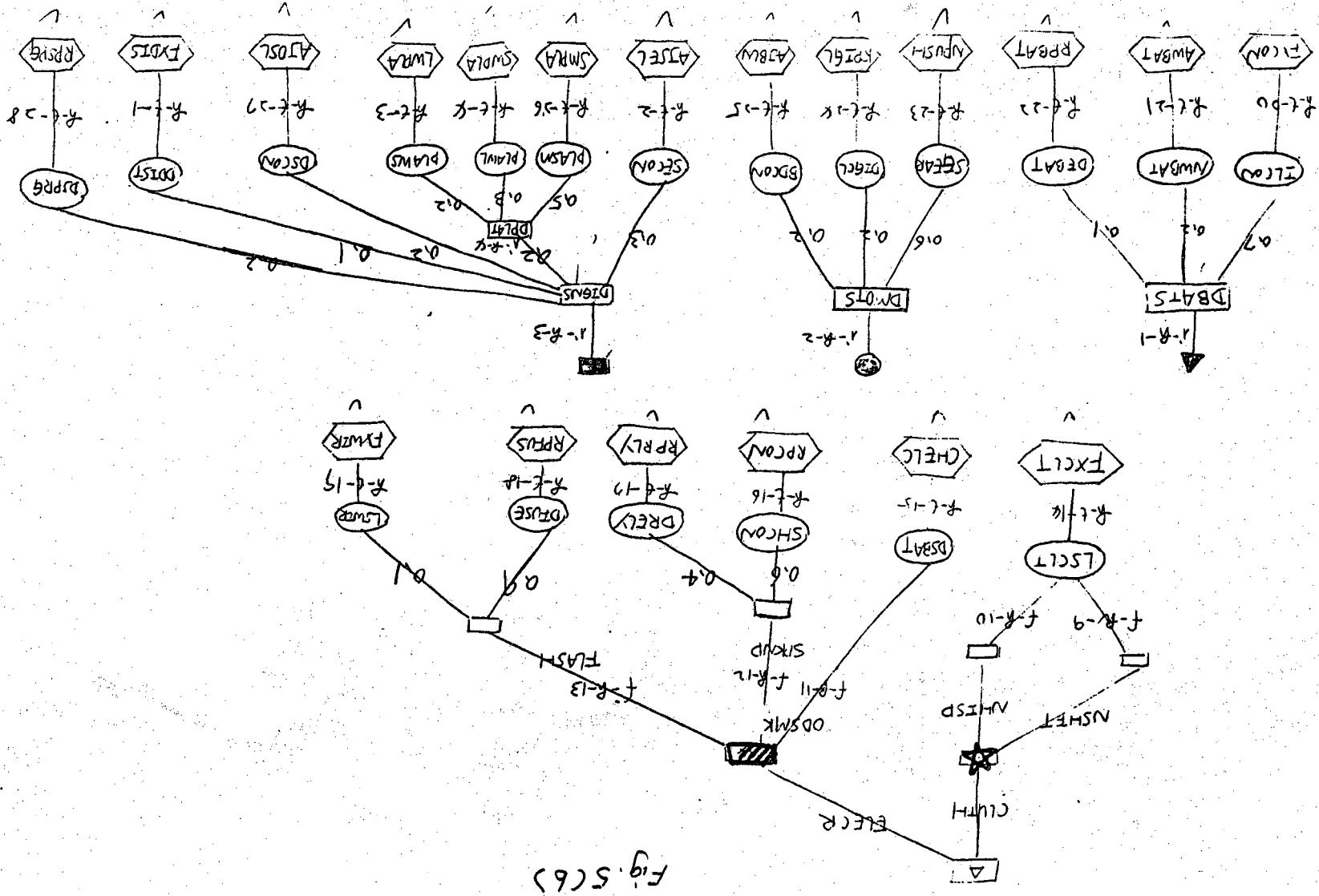


Fig. 5 (b)

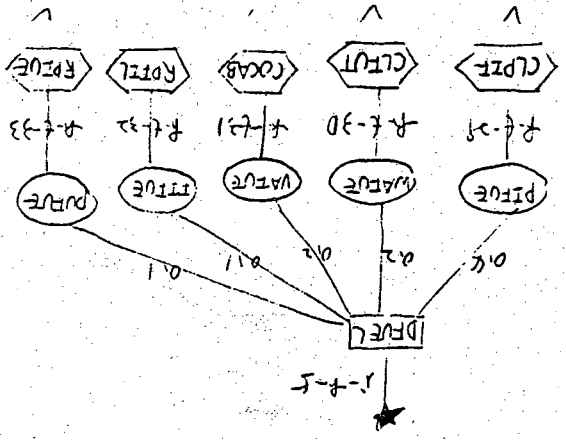
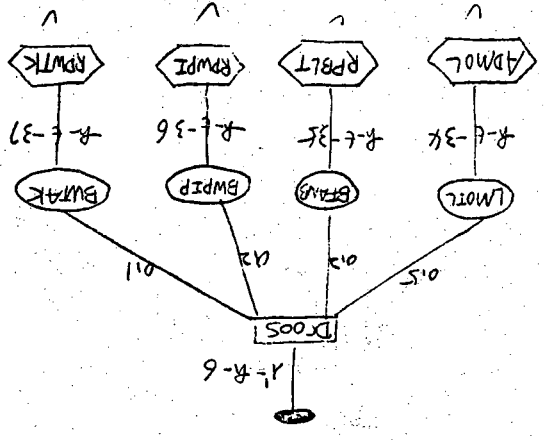
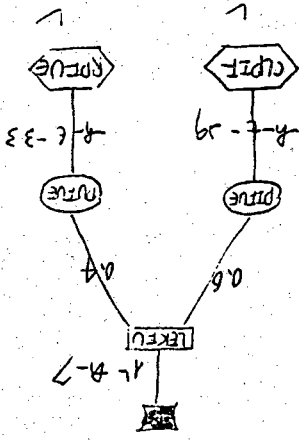
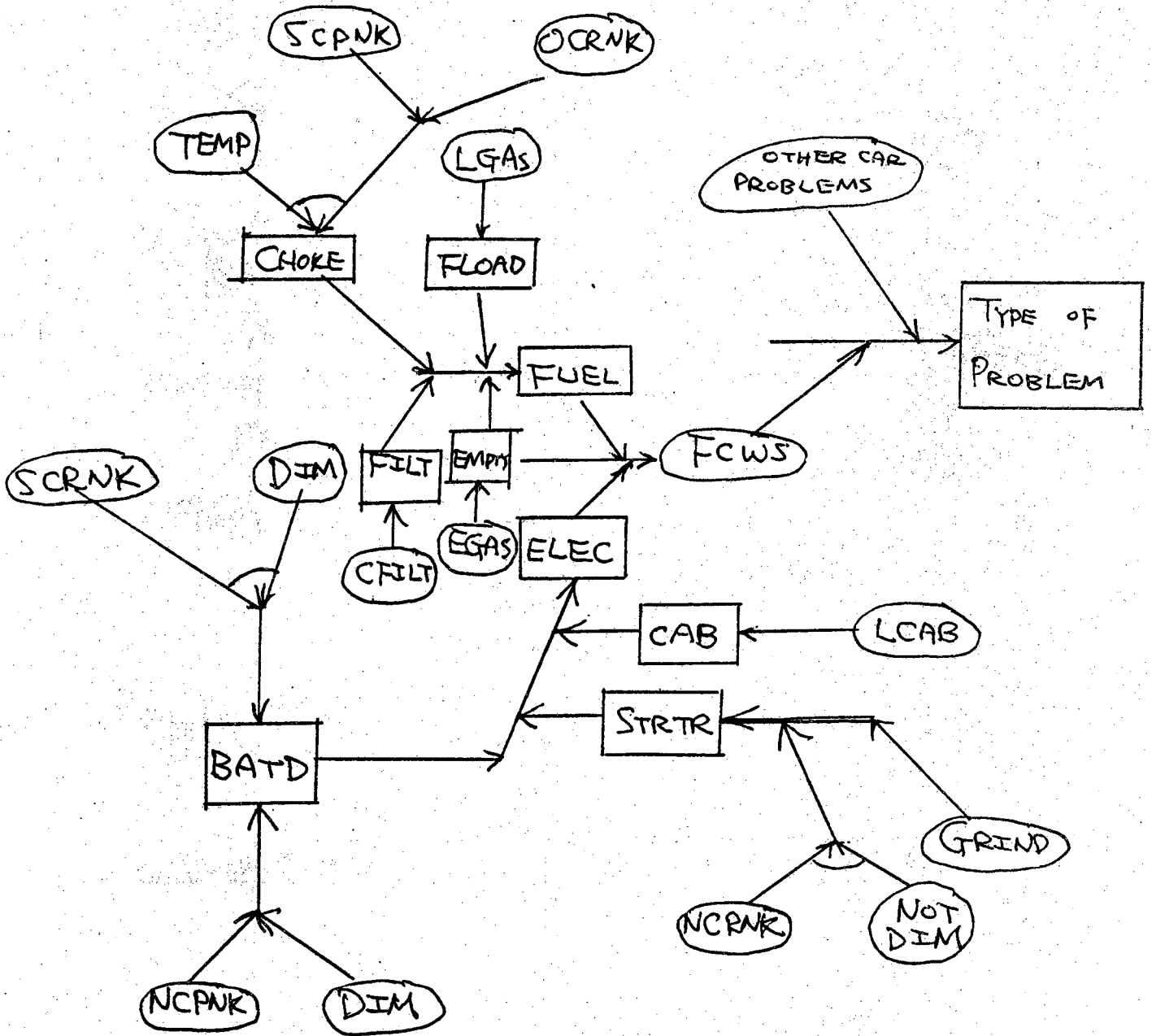


Fig. 5(c)

Rule = f - x = 6  
 f - x = 13  
 x - x = 7  
 x - x = 37



LEGEND: ○ = FINDING  
 □ = HYPOTHESIS

Figure 6. Cause-and-Effect Diagram of Weiss's Car Repair Model.

Table 2. Explanation of the Abbreviated Variables in Weiss's Car Repair Model.

(BATD : battery discharged)  
(CAB : battery cables loose or corroded)  
(CFILT : fuel filter clogged)  
(CHOKE : choke stuck)  
(CLEAN : clean and tighten battery cables)  
(DIM : headlights are dim)  
(EGAS : gas gauge reads empty)  
(ELEC : electrical system problems)  
(EMPTY : no fuel)  
(FCWS : car won't start)  
(FILT : fuel filter clogged)  
(FLOOD : car flooded)  
(FOTH : other car problems)  
(FUEL : fuel system problems)  
(GAS : put more gasoline into tank)  
(GBATT : charge or replace battery)  
(GRIND : grinding noise from starter)  
(LCAB : battery cables loose or corroded)  
(LGAS : very strong odor of gasoline in carburetor)  
(MGAS : normal odor of gasoline in carburetor)  
(NCRNK : no cranking)  
(NGAS : no odor of gasoline in carburetor)  
(NSTAR : replace starter)  
(OCRNK : normal cranking)  
(OPEN : remove air cleaner assembly and manually open choke+ with pencil)  
(RFILT : replace gas filter)  
(SCRNK : slow cranking)  
(STRTR : starter malfunction)  
(TEMP : outdoor temperature(degrees F))  
(WAIT : wait 10 minutes or depress accelerator to floor+)

1/26

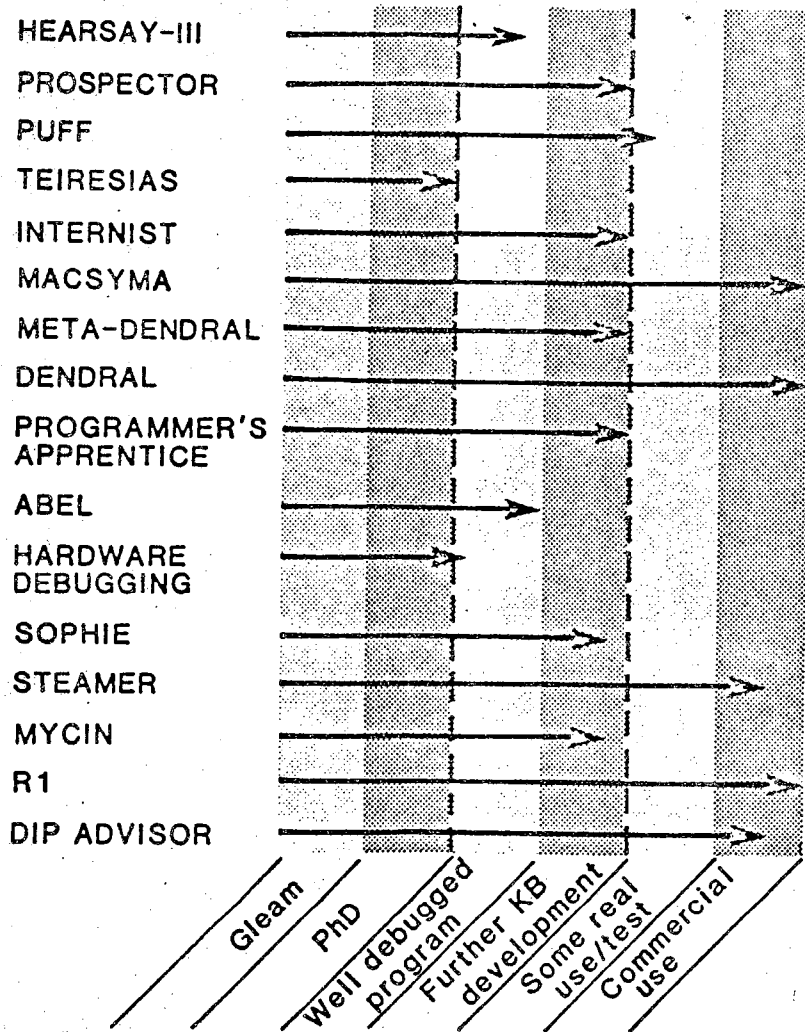
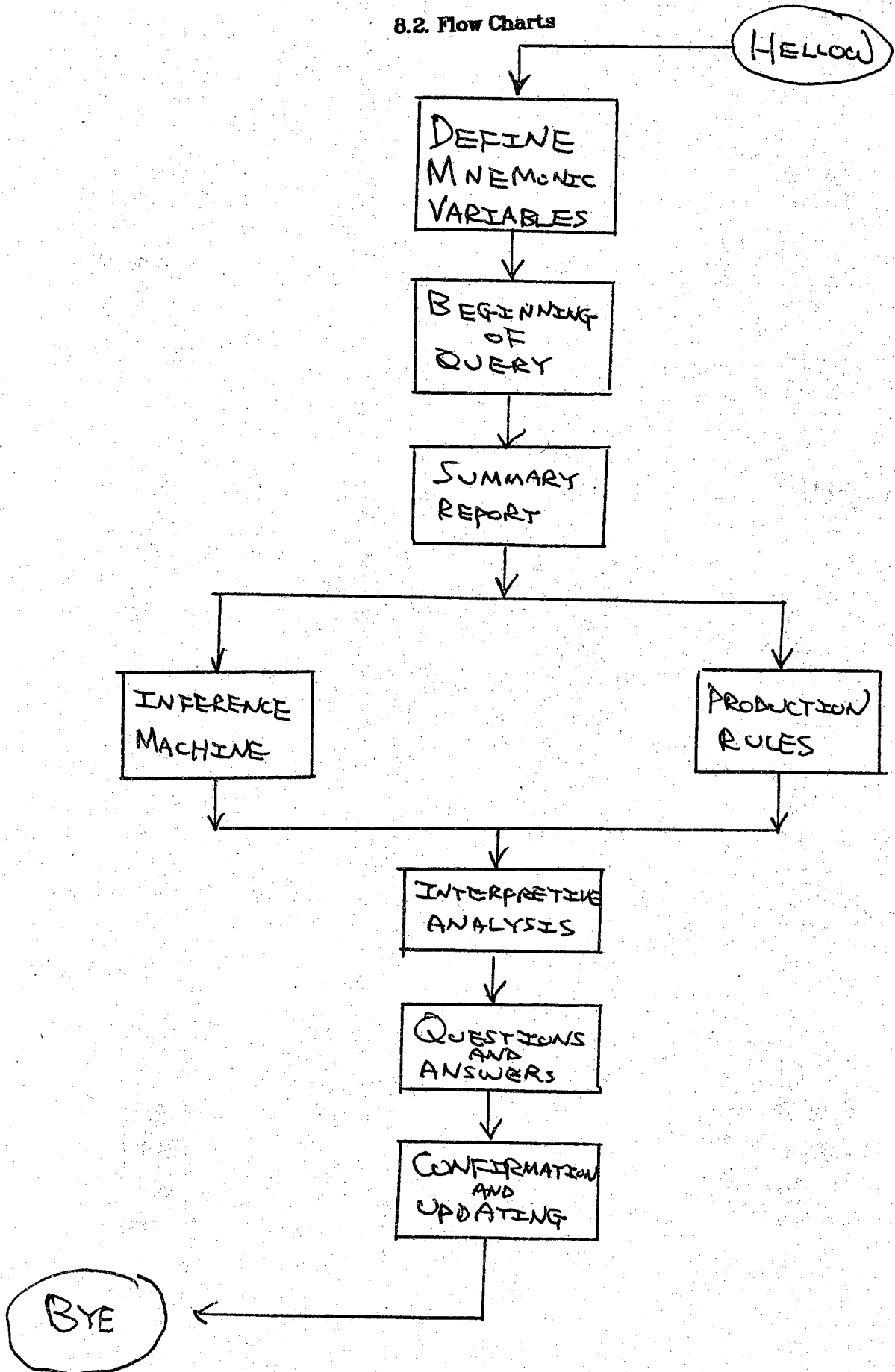


Figure 7. State of Development of Some Well-Known Expert Systems.

### 8.2. Flow Charts





---

**Chapter 4**

**Deep Drawing Feasibility Expert System**

*G. Eshel*

# Deep Drawing Feasibility Expert System

- 167 -

Gad Eshel

## ABSTRACT

Deep Drawing Feasibility Expert System ( DDFE ) checks producibility of axi-symmetric parts by the deep\_drawing processes. The system will be incorporated within a general metal\_working process\_planning system, that will include forming processes, for generating multi-technology process\_outlines.

Before entering the detailed-specifications design stage of a process, the process\_planner checks the feasibility of producing the specified part by the examined process. The expert knows the capabilities of the process he is considered an expert in, the significant features of the raw\_material, and the resources available - raw\_material in stock and the capability of the machines.

The DDFE is designated to be operated by the designer or by the process\_planning coordinator. Both are not domain experts ( in the deep-drawing technology ). Nonetheless, they seek an answer about their feasibility inquiry. That's why the system is not interactive, - it goes by the fundamental principle of automatic process\_planning, as opposed to computer aided process\_planning.

Since a complete theoretical analysis of the deep\_drawing process has not yet been accomplished, the expert is vital. To infer the answer the expert uses a combination of theoretical computations, empirical results, and approximate measures. Extracting the process-capability knowledge, and putting it into a formal representation, is the core of the DDFE, and the most challenging task of the system designer. The essentials of the process\_planning approach and the deep\_drawing process are described in the first part of the report.

The knowledge of the DDFE is organized as a hierarchical rule\_based system. The data\_base and the inference machine, are largely independent and user\_updatable. Inference, is done through primitive matching of Horn clauses. It draws primarily on the built-in theorem-proving capabilities of PROLOG, - the language it is programmed in - and higher level, "macro"s. To the extent possible, data\_base ( relational, represented as: " PROLOG facts " ) is generalized in frames, and rules are structured. This facilitates almost unrestricted data\_base updates and, some rule modification capability.

The portion of the DDFE described hereby gets as an input, the CAM representation of the required part, and tests producibility, employing the main process ( "cupping" ) and redrawing.

It works and has produced sound advices.

1. TERMINOLOGY

General:

The evaluated workpiece is represented by the right-hand side, of the cross\_sectional view, of a plane through the axis of symmetry.

Sizes: longitudinal sizes given in inches, angles : degrees.

Abbreviations in the program:

H - denotes height, or, in list processing; Head of a list.

Final-shape - in program: the shape at end of process.

ID - inside diameter.

Initial-shape - in program: the shape at start of process.

OD - outside diameter.

R - denotes : radius. May be prefixed or suffixed to another term.

RM - raw\_material.

T - in list processing, denotes Tail of a list.

WT - wall thickness, default for: nominal wall thickness.

Notation for flow diagrams:

FF - form feature,

SE - Shape Element,

WP - Workpiece.

LEGEND:

|  
----->----- - a possible path for material flow

"""""""""" - a possible path for process\_planning

~~~~~ - a cut in the global picture, to isolate a window.

x - a fork: path selection.

(yy) - the No. of the conceptual process\_planning operation.



mented.

DeMachine, DeForm, DeDraw, ... - the conceptual backwards, process\_planning activity of producing the initial shape of the workpiece, once its final shape in the particular process is given.

Deep Drawing Ratio - deep drawing ratio. Usually denotes the maximal feasible cupping for a complete draw. Defined as:  $\frac{OD_{RM}}{OD_{cup}}$

design - assumed to conform with principles of design-for-manufacture.

envelope - the wall + the two bases.

fabrication - workpieces joined so that to disassemble them means destruction of the components.

Feasibility - implies both technological workability, availability of resources, and an acceptable, upper bound limit, to the effort to achieve it. This last measure would be referred to as: "generalized cost". Further elaboration in & 2.

finished form - ( or: finished shape ) the shape and mechanical properties of the material after the end of the worked process. Note: "finished form" pertains to any intermediate process, while "finished required part" pertains to the final process.

finishing processes - processes which do not change the nominal sizes but the surface texture.

initial form - ( or: initial shape ) the shape and specifications

of the material, of the workpiece, at the start of the process.

main shape - the "thinned" ( skeletonized ) outline of the envelope, - where its form features are "filled" with material.

main shape of a process - the standard prototype shape of either the initial or the final shape of a process.

massive Forming processes - A Forming process in which the shape undergoes massive change, usually Hot-Forming, and IS is a Billet.

methodologically - ( or: symbolically ) applies to a demonstration that is included schematically or by an example only, to demonstrate existence and to imply that a thorough work is needed to include the entire scope. It is assumed that this form of presentation does not cause loss of generality of the issue at stake.

net shape-forming - forming process that yields a final shape and a surface that does not require finishing. Usually, this notion pertains to forging processes.

parametrization of a process - a very flexible definition for filling up the frame of a process with semantics.

preform - a raw shape produced by a massive forming process.

primary processes - massive forming + casting .

primary workpiece - a one-component workpiece - there is no joining operation in the current context of evaluated manufacturing processes.

priority of a process - the notion is used here when several processes compete over producing a certain feature, to denote that a certain process is always preferred to another.

process outline - the twin-tuple sequence of name\_of\_process and shape\_of\_workpiece at beginning of process.

process plan - the sequence of the processes and process parameters needed to produce a defined part to design specifications. Does not include control instructions of the equipment actuators.

process planning - the process of determining the process plan.

prototype shape - the type of shape that a process is capable of producing ( might be referred to as : "simple prototype shape", too ).

R-value - the ratio between the plastic strain in the stretching direction ( width and/or length, or hoop stretch ) to the ration in the orthogonal direction ( wall thickness ).

secondary processes - processes that employ a relatively small deformation ( drawing, spinning, swaging, ... ) or significant metal removal ( to exclude various finishing processes ).

semi-composite process - a sequence of a simple process followed by supplementary processes.

shape feature - a FF that is produced, clearly and solely, by a sheet metal process.

simple process - a secondary process whose initial shape is stock

material.

stage - ( or: pass ) usually pertains to forming processes: an element of the forming process which performs a change of shape. Corresponds to a "cut" in the machining process.

sub-assembly - a lower level assembly. can be a fabrication or a workpiece.

supplementary process - a forming process that modifies a certain shape element of the entire workpiece and thus complements the main process.

technology - a sequence of main processes ( each might be a multi-stage ) and aux. operations to work a given initial form into a finished product.

wall - the rotational profile of the workpiece, - without the bases.

workpiece - a primary mechanical part while in the process of being worked on.



2. ENVIRONMENT, AUTOMATIC PROCESS PLANNING, DRAWING PROCESSES and THE ROLE OF THE EXPERT

2.1 INDUSTRIAL ENVIRONMENT, SCOPE

Environment.

The industrial environment the DDFE is incorporated in, is the small\_batch, one of a kind, technologically advanced - manufacturing. The prevalence of this mode is widely publicized and will not be elaborated here. Normally, in this manufacturing environment, whenever a process\_plan for a workpiece is designed, candidate processes to compose the processes\_outline, unto which, the part will be manufactured, are all the known processes in the plant/enterprise. Thus, expanding the Automated Process\_Planning Methodology, to include non-chip-forming processes is a real industrial necessity.

Families Of Products And Processes:

The "holistic" idea is demonstrated through a simplified family of parts and processes. The parts are workpieces with rotational symmetry. This property leads to the use of rotational-shape-producing processes. A domain that contains machining, as well as various forming processes, like: drawing, spinning, tube\_sinking, and raw\_tube-producing processes. An expansion can include special surface deformation processes, which are grouped into the finishing processes ( like shot-peening, burnishing, etc. ) too. The system is currently capable of generating deep-drawing processes, only.

A process is capable of producing simple prototype end\_shapes out of a predefined raw\_workpiece prototype, ( see: prototype ) only. This constraint may be later released.

Types of materials, forms, and sizes of raw\_material are the commercially available ones. "Commercially available" means here: within the range of products, of the raw\_stock producing mills, - not confined only to stock that is held usually in warehouses, or considered an 'off the shelf' product.

Comment: Axisymmetric Products.

The family of rotational parts constitutes a major share of the batch manufacturing industry, and thus, the need to generalize the scope of products, is not that acute. - Due to its large share of the above described industry, furnishing a solution to the family of rotational parts is of great significance, even without applying the methodology to other types of products.

Representation of technology:

Technology is represented realistically : boundaries will be set for normal industrial use, and not for the special cases. Assumptions, with regard to processes and relationships among the processes, apply to the bulk of the products in this environment, but not necessarily, to 100% of them. ( e.g.: Usually, the spun surfaces in the tube-shear-spinning process are not machined, although they might on rare occasions. A rule that states : "spun surfaces are not machined" covers the normal shop practice ).

## 2.2 FORMING PROCESSES WITHIN THE MANUFACTURING PROCESSES.

### 2.2.1 The Plastic Deformation

A solid body subjected to a force of small magnitude is deformed elastically such that the strain is directly proportional to the stress. When relieved of the stress, the body returns to its original dimensions. Therefore, elastic deformation is a reversible or recoverable process. But, when the imparted forces result in a composition of stresses that exceed a certain yield boundary, the body is permanently deformed, and the process is irreversible, or: irrevocable. Actually, there is no complete regain of the deformed strain in elastic deformations, and, on the other hand, each irrevocable process exhibits a small rate of elasticity. But, definite revocability / irrevocability are useful idealizations, even though a range of elasto-plastic problems is treated in plasticity.

The main properties that determine the workpiece behavior under stress, vary extremely with regard to material, its grain structure, temperature, duration of the process, the history of deformations taken place.

The theoretical theory of plasticity attempts to formalize experimental observations of the macroscopic behavior of a plastically deformed workpiece.

To put in the appropriate context: the physical explanation of the elastic and plastic properties of solids, is taken from the microscopic point of view, - in relation to the material crystal structure.

A comprehensive study of plasticity in [ Johnson, Slater ].

The term: "Forming processes" pertains to the whole range of processes in which the shape and mechanical properties of a solid metal workpiece, are altered without material removal. In spite of the remarkable advance of knowledge about the process during plastic deformation, and development of new analysis methodologies, in the last decades, full knowledge is yet illusive, and forming processes expertise remain largely experience based.

### 2.2.2 Forming Processes, and Their Classification

There is no accepted classification method for the enormous number, and variations, of forming processes utilised in industry. The more common ones are:

- a. characterization by the homologous temperature - hot, warm, cold.
- b. chip\_forming vs. chipless forming ( already refered to above ).
- c. a mechanical analysis point of view: state of stress of the workpiece ( simple, complex, uniaxial, .. ).

- d. a type\_of\_stress\_involved view: tensile, compressive,...
- e. characteristics of plastically deformed zone: size, local ( sheet forming ) <--> comprehensive ( bulk deformation processes ).
- f. amount of strain rate involved.

Several methods tried to unify the advantages of each of the classifications above. Thomsen, Yang and Kobayashi [ Slater ] suggested a scheme based upon the kind of stresses involved, taking into account their properties. They have ended up in 4 groups, which will serve as a reference for further classification in the course of this work.

1. Squeezing group: the workpiece is subjected to compressive stresses and a large change in shape is produced. - forging, extrusion, rolling, swaging, spin and roll forging. Most of these processes are hot worked.
2. Drawing group: the workpiece is subject principally to a tensile stress. Generally, a smaller plastic deformation is achieved per operation. The workpiece is usually a bar sheet or tube, and primary changes are in shape, while changes in thickness are the result of those changes. - processes in this group include: wire bar and tube drawing, and the range of deep drawing processes.
3. Bending group: here the workpiece is subjected to couples, thus inducing stress gradient throughout the thickness. Change of shape is dominant, while change of thickness is mostly of secondary order. - flanging and break-form operations are included in this group.
4. Cutting group: chipless forming: - piercing, blanking, shearing, ... Chip forming group: conventional machining operations. A third subgroup include more recent processes, mainly small-rate metal removal processes: EDM, ECM, ultrasonic machining, electron-beam and laser machining.

Sheet metal forming is a broad term for metal-working processes in which the shape of a punch and a die is reproduced directly in the metal. That's why, it is sometimes called: " press forming ", or: " die forming ". From the kind\_of\_stress classification all the sheet metal processes fall within the drawing group.

Another classification of metal working processes, which is found useful for embedding the sheet metal forming processes, differentiates between primary and secondary processes, while the secondary are categorized by their ( surface / volume ) ratio. The primary processes impart large strains, shape fluid metal or mold doughy metal. This group corresponds to the " Squeezing group " + casting and powder metallurgy.

The secondary processes group include, thus the following families and processes ( U-S designates, here: Uni-Stage operation process, the default is multy-stage ) :

Forming:

Low surface/volume

- main processes:
  - \* Tube-Drawing (with stationary / floating mandrel)
  - \* Shear Forming ( tube spinning, cone spinning )
  - \* Deep Drawing ( ironing, tube sinking )
- supplementary ( to complete/modify shape )
  - \* Nosing ( Tube Sinking, Tube Expanding ), ( U\_S )

Large surface/volume

- main processes
  - \* Deep Drawing ( rubber forming, hydroforming, with/without blank holder )
  - \* Spinning (forming various surfaces of revolution)
  - \* Roll Forming + Weld., ( U\_S )
- supplementary ( to complete/modify shape )
  - \* Dimpling ( radius, cone, flange ), ( U\_S )

tubular shape\_producing methods

(with some overlap with above classified processes):

- \* Shear Forming: Tube Spinning
- \* Ring Rolling, ( U\_S )
- \* Roll-Forming + Weld., ( U\_S )

Metal removal:

- large scale : shearing
- small scale : point - machining.

Note:

the various plastic surface deformation processes ( shot and blow peening, burnishing, etc. ) are grouped into the finishing processes and therefore will not be included, although the need to utilize a process of this family will be stated.

## 2.3 GENERALIZATION OF AUTOMATIC PROCESS-PLANNING.

### 2.3.1 Process Planning Functions & Phases:

Process\_planning is a function and a process. The function is: " The subsystem responsible for conversion of design data to work instructions ", [ Weil ].

This view of process\_planning is not confined to the metal removal domain of processes, although, currently, it is a standard practice to delimitate it to machining only [ Weil ].

While production planning is intended to realize the efficient utilization of resources to obtain production goals, process\_planning is exclusively concerned with the selection of the processes and process parameters to perform the manufacturing operation itself.

Process\_planning ( manual or automatic ) can be viewed as a tree of activities . In the process of determining the methods and the sequence of processes to work out a required finished component, to design specifications, 13 main phases can be distinguished:

1. Selection of the process\_outline.
2. Determination of raw\_material allowable forms and conditions.
3. Determination of auxiliary operations, within the process\_outline framework.
4. Determination of elements within the each process of the process\_outline.
5. Tool / die design.
6. Machines selection.
7. Determination of holding / clamping devices.
8. Determination of inspection instrumentation and tools.
9. Determination of the intermediate shapes and tolerances.
10. Parameterizing each operation / element. ( e.g.: cutting conditions, press speed, ... ).
11. Identifying difficult / impossible design features and correct them, with designer, if possible.
12. Working duration, - direct and indirect.

---

1. extending [ Weil ] version of a machining\_only process\_planning.

13. Selection of the optimal / " best " for shop needs process\_plan.

The DDFE is going to concentrate on the closer\_to\_the\_root branches of this tree of activities:

- selection of raw-material / initial form
- selection of a technology ( the sequence of processes )
- specifying initial and final form at the end of each operation.
- assigning machines to operations.

2.3.2 Foreward <--> Backward Process Planning

Before an activity is automated it must be well formulated in a formal way. Process\_planning automation is difficult because it is not a straight forward activity. The manufacturing processes, on the other hand, are much easier formalizable. The manufacturing activities have definite order and procedures to be carried out upon. they consists of discrete sequential steps. Defining the process\_planning activities in terms of the manufacturing operations, yields them formalizable too. One useful, recent, method for formulating the process\_planning activities, in terms of the manufacturing activities, is the backward process planning or: inverse-manufacturing. By this approach, every process\_planning activity is seen as the inverse of the manufacturing activity, along the whole ladder of the process\_plan.

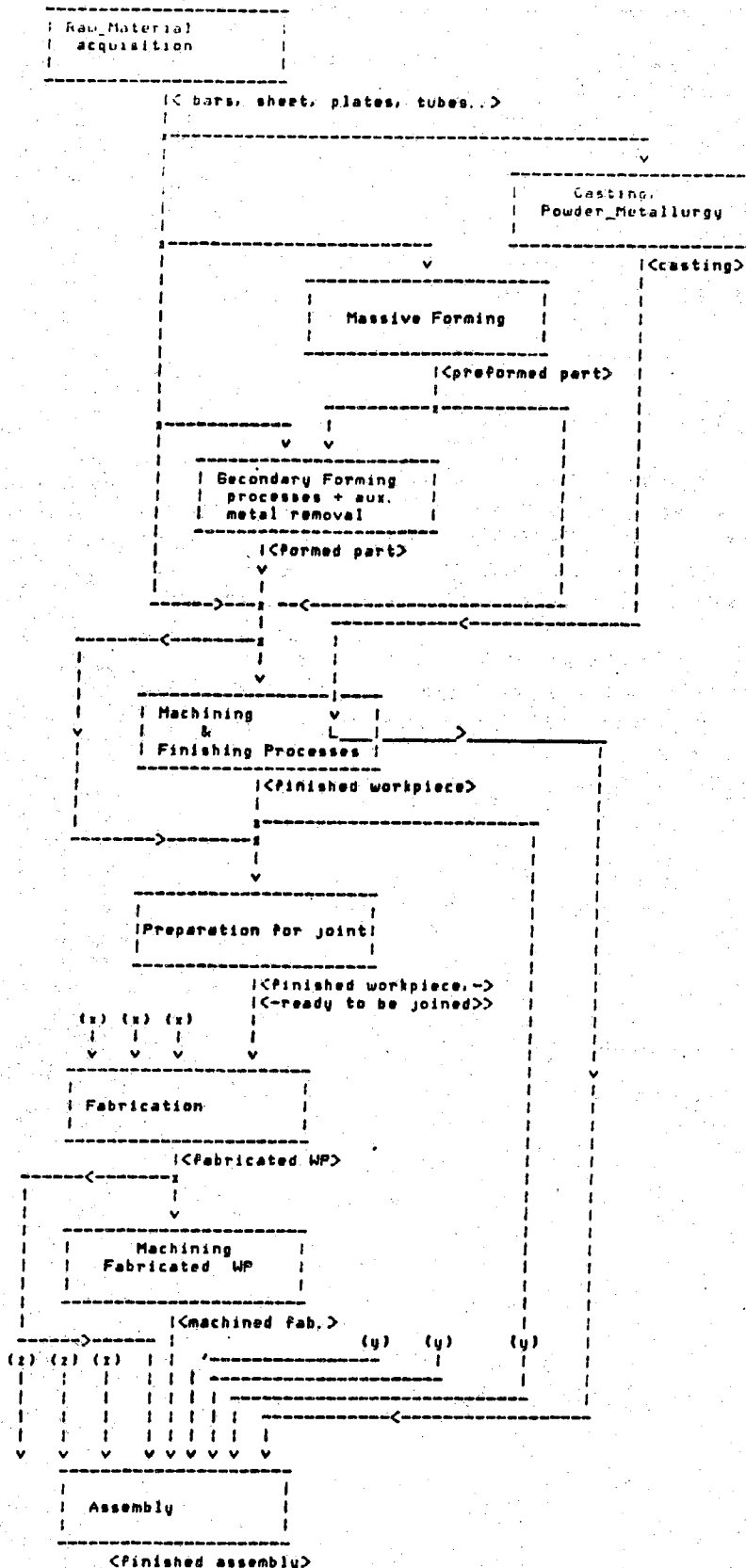
Applying this principle, for drilling a hole, would mean: fill up the tubes removed during the machining activity, starting with the finished hole, ending with the hole filled to its initial-state-surface. Similarly, for a forming process, instead of drawing the cup from a blank, the inverse process\_planning would mean: straighten the cup to produce a blank, such that can be found in stock.

This approach, while implemented to the sequence of operations that comprise the process\_plan, means: a backward sequence of processes, starting from the finishing process, ending up with the raw\_material recognised in stock.

2.3.3 Generalizing The Domain Of Processes For Process Planning.

As noted above, expanding the applicability of process planning beyond the chip-forming processes, is one of the primary objectives of building this expert system. One view of such a generalized system, is described here by.

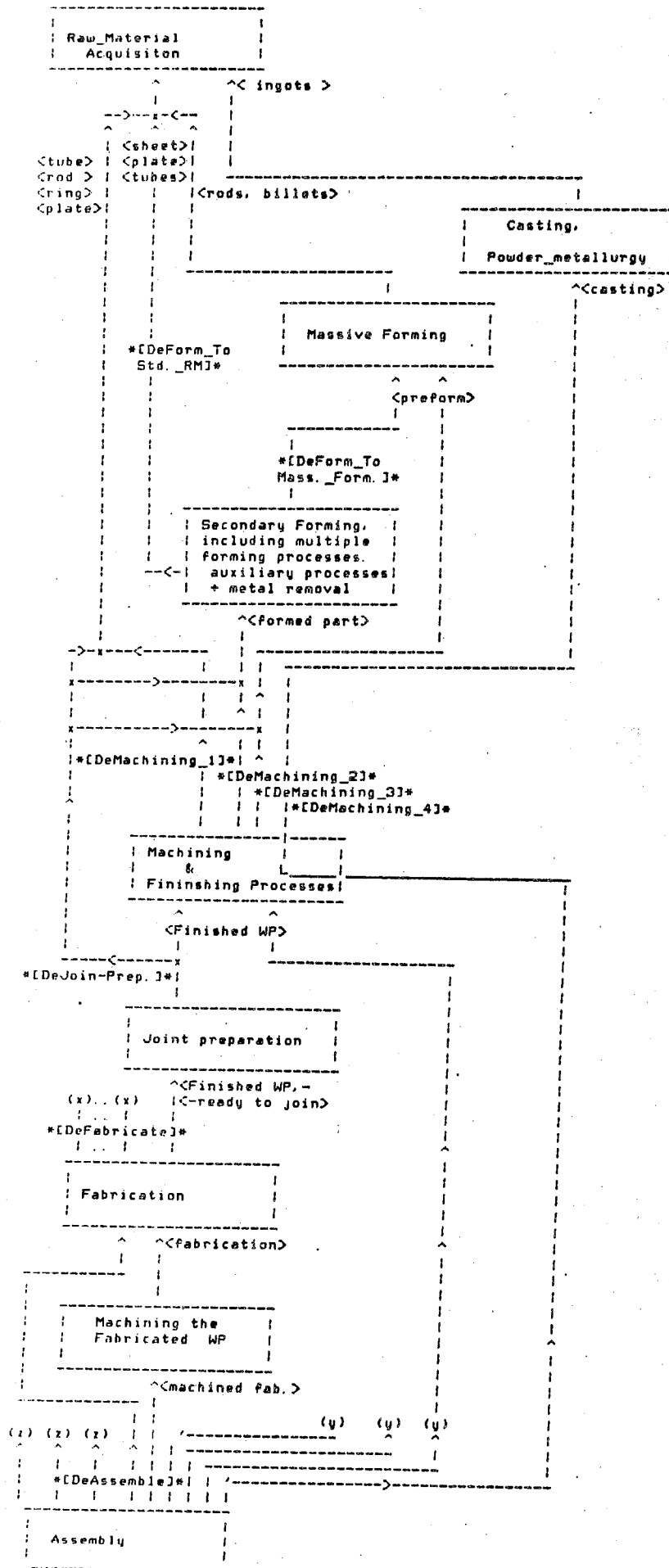
MATERIAL FLOW IN THE MANUFACTURING ENVIRONMENT:



Note: Sheet metal parts, that may employ several forming processes following a large-scale MR process, are all included in the domain of: 'secondary forming processes + aux. metal removal'.



PROCESS PLANNING ANALYSIS ELOH



## 2.4 ESSENTIALS OF THE DEEP DRAWING PROCESS.

Sheet-metal-forming processes, broadly referred to as: "press-forming", "die-forming", "deep-drawing" or: "stamping" processes, represent a wide spectrum of flow conditions. At one end of the spectrum, stands the forming of a flat bottomed cylindrical cups out of a flat blank, ( cupping ), where principal strain is positive ( tensile ), the other is negative ( compressive ), and the change of thickness is negligible. At the other end we find biaxial-stretching operations where two of the principal stresses are tensile, and resultant thinning is significant.

Sheet metal forming is distinguished from bulk-forming processes, in that here, tension dominates, and one or more surfaces of the deformed region are not supported by tools.

Formability depends upon: lubrication, tooling, rate of material flow, material properties, and true slip\_lines ( the lines in the deformation zone along which shear occurs ). The slip\_lines depends largely on the radii of the boundaries connecting the zones of flow. See [ Johnson ], [ Koistinen ], [ Niemeier ].

A simple illustration of the relatively simple process : cupping, would explain the some of the basic highlights of the metal flow in the deep drawing processes. [ Johnson ]. The essentials of the tools are shown in fig. 2.4.1, and the progressive states of the drawn blank illustrated in fig. 2.4.2.

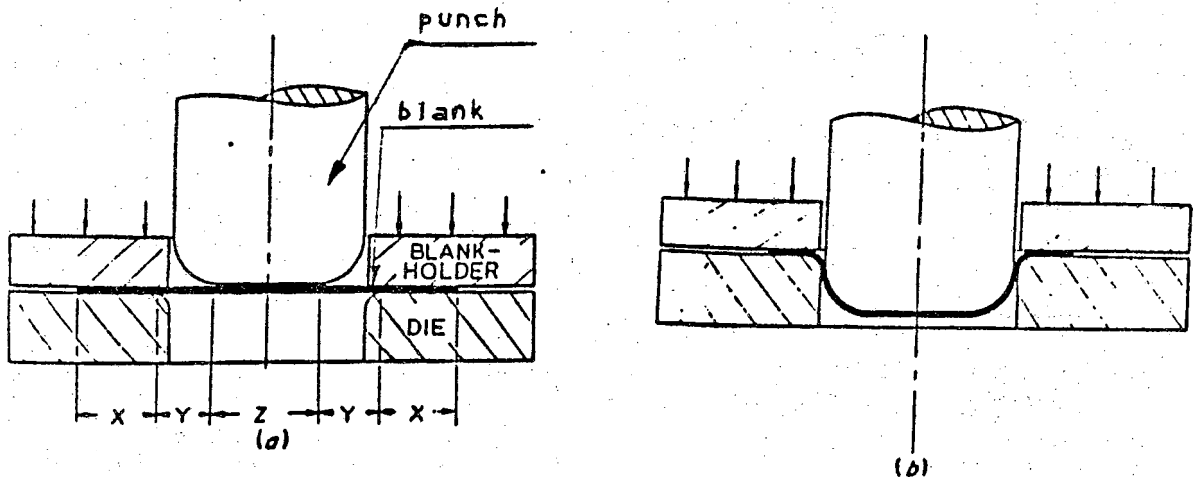


fig. 2.4.1 - tools in cup drawing.  
( from: [ Johnson ]. )

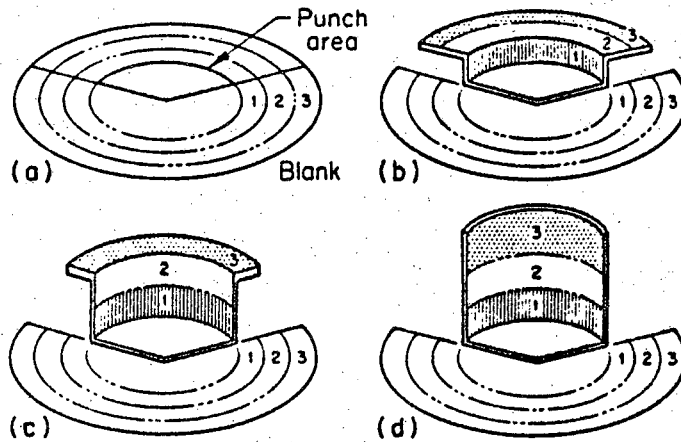


fig. 2.4.2 - progressive states of drawn blank.  
( from: [ Lyman ]. )

For a more extensive evaluation of the drawing process, - see [ Woo, Slater, Johnson, Lee ].

Analysis of the cupping process shows 5 distinct regions developing in the blank [ Johnson, Slater distinguishes 6 zones ], - see fig. 2.4.3 .

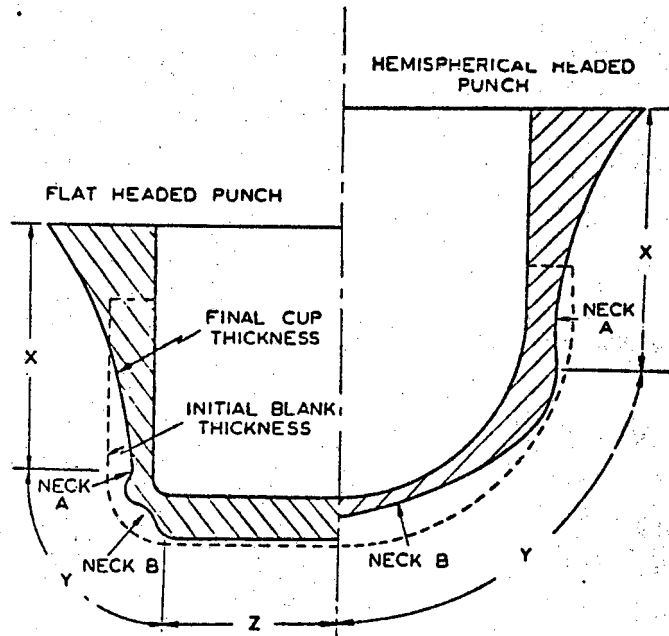


fig 2.4.3 - sections during cupping.  
( from: [ Johnson ]. )

A forming limit in this process occurs when the stress at one of the local necks, drawn in fig. 2.4.3 exceeds the yield stress, - then the local necking would start to elongate in the direction of tension, until fracture reached. The most frequent defects are: wrinkling, puckering, exaggerated earing, wall tearing, and edge cracking. An extensive discussion of defects common to the deep-drawing family of process in [ Johnson2 ]. Some typical deep\_drawing defects in fig. 2.4.5.

Sheet metal forming is limited either by wrinkling or buckling of the sheet, due to inadequate restraint or insufficient tension, or, by tearing, out of excessive tension. From the analytic point of view, failure here, is usually caused by plastic instability in tension, rather than fracture. Nonetheless neither of the two most common measures of tensile ductility: - reduction in area and total elongation, - correlate significantly with the process performance. For each process, material, and operational parameters; if a large number of tears are examined, a curve which establish the "secure" boundaries of strain, can be drawn. This curve is called: Forming Limit Diagram, it has the typical shape as in fig. 2.4.4.

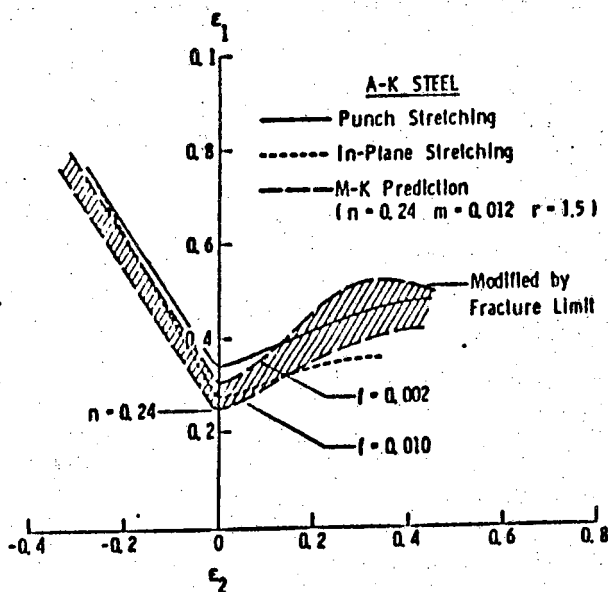
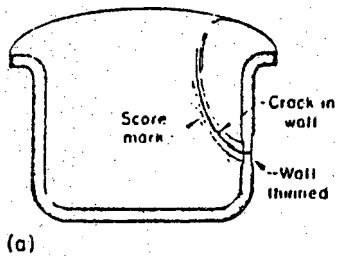
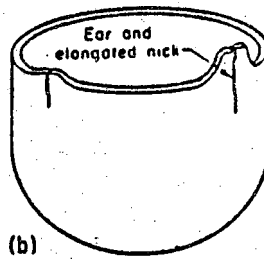


fig. 2.4.4. - typical forming limit diagram  
( from: [ Ghosh ]. )

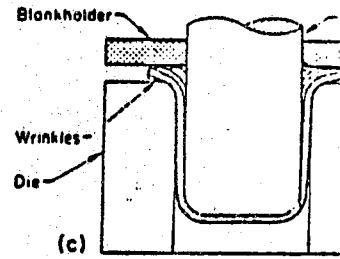
See [ Andersen ], [ Koistinen ], [ Hecker ], for more details.



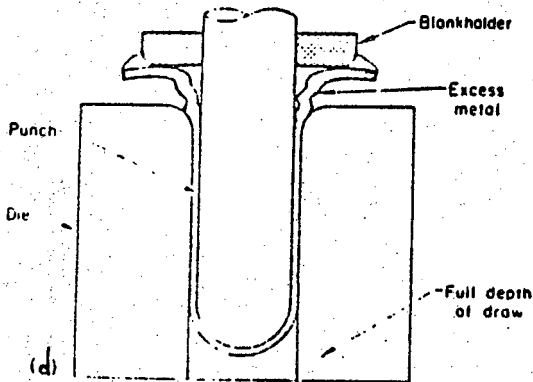
wall thinning



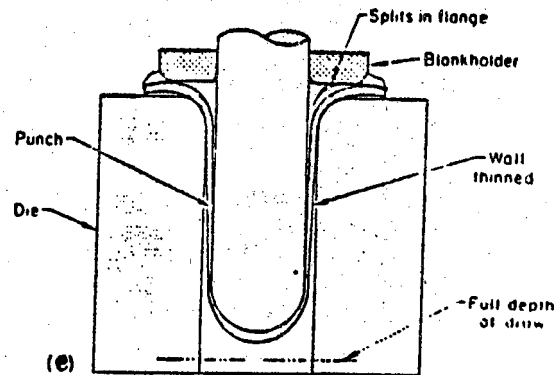
earing



wrinkling



puckering



composite problem

fig. 2.4.5 - some typical defects occurring in deep drawing.  
( from: [ Lyman ]. )

## 2.5 COMPLEMENTARY DEEP DRAWING PROCESSES.

When the basic preliminary draw can not produce the required cup, additional complementary processes are required. The most common ones in practical die-forming industry are:

- i. Redrawing,
- ii. Sizing : Flanging,  
          Contouring,
- iii. Reducing: Necking,  
          Nosing,
- iv. Expanding: Regular,  
          Bulging,
- v. Ironing,
- vi. Embossing.
- vii. Tractrix\_die drawing
- viii. Hydrostatic drawing

The main process itself, can be categorized, by the type of strains developing during the draw, into three main groups:

- i. cylindrical\_cupping,
- ii. taper\_cupping,
- iii. spherical\_cupping,

Analytical insufficiency does not prevent the deep-drawing processes from being carried out extensively in the industrial world. [ Wick ], [ Lyman ]. The inability to provide exact solutions has led to the development of empirical knowledge and approximate rules. [ Hobbs ]. These rules mostly rely upon consequential observations of the drawn shells. Such knowledge furnishes in practical performance; formability limits, blank holding force and the relationships between the various radii ( bottom fillet, flange fillet ), of the drawn shell. The subject of formability limits has only recently been extensively researched, thus, furnishing a more complete practical know-how. [ Pittman ].

## 2.6 OTHER AXI-SYMMETRIC SHAPE PRODUCING PROCESSES

As noted above, the cause of automatically generating the process\_outline is the existence of competing, axi-symmetric\_shape producing processes. Some of the outstanding ones are:

1. Hydroforming:
  - i. Guerin / Verson\_Wheelon / Marform,
  - ii. Rubber\_pad\_forming,
  - iii. Bulging.
2. Spinning
  - i. Metal spinning
  - ii. Flow forming
  - iii. Shear forming.
3. Explosive forming:
  - i. Thin\_shells,
  - ii. Thick\_shells.

each of these processes, has its advantages, and shortcomings, especially, with regard to range of sheets and materials worked, and the outcoming mechanical properties. As a first stage extension of the system, it is recommended, to embed these processes.

## 2.7 THE ROLE AND THE NEED FOR OF THE DRAWING-PROCESSES EXPERT

The role of the metal\_forming specialist ( expert ) is to provide the designer and the manufacturing engineers with information required to design the product and efficiently operate a metal-forming process. The information includes the nature and extent of the deformation involved, and the forming parameters to facilitate the process, such as: force, power, lubrication. As indicated above, no complete analytical solution of any of the metal forming processes, has yet been accomplished. Among the main difficulties in obtaining such a solution:

- i. The phenomenological nature of the mechanical properties of the materials. We are able, thus far, to characterize the material behaviour, in various ranges, but not to deduce quantitatively its properties from its structure.
- ii. Material properties are constantly changing during the flow of the metal. Its dependence upon the temperature, rate of flow, and internal flow directions has not yet been fully understood.
- iii. Plastic anisotropy ( inhomogeneous features of different directions ) and the exact values of friction, at each zone of deformation, are neither constant nor known.
- iv. A complete solution, which involves elasticity and the changing conditions of plasticity, is mathematically complex.

Because of the analytical complexity, process\_planning of forming processes is done by specific\_process experts. It is a common industrial practice to rely upon different experts for each sheet metal process, and even for different types and sizes of products. Moreover, specialization and expertise, are confined to class of products. Thus, one can find that conferences for producing stampings for the auto industry and for the airplane-building industry deal with different material. [ Koistinen ], [ Chen ].



2.8 EMBEDDING THE DRAWING-PROCESSES EXPERT WITHIN A RULE BASED SYSTEM FOR GENERATING AUTOMATICALLY COMPOSITE PROCESS OUTLINES.

One possible incorporation of the DDFE in a more comprehensive, generalized, process planning system, that includes other axis-symmetric shape producing processes of class-II ( drawing-stresses ), is shown below:

( for legend: see & terminology ).

. COMPOSITE PROCESS

INPUT TO PROCESS PLANNING:  
<Finished WP>

-----  
| Machine : |  
|-----|  
| Nonrotational FF's |  
|-----|

|<Finished WP, with material fill->  
|<for nonrotational FF's>

(1)!\*[DeMachining nonrotational FF's ]\*  
v

-----  
| Machine: |  
|-----|  
| Rotational FF's |  
|-----|

|<Finished WP, with material fill->  
|<for rotational & nonrotational FF's>

(2)!\*[DeMachining rotational FF's ]\*  
| (The FF's that can be obtained by machining only)

-----  
| Machine : |  
| complement FF's |  
| of Forming Proc. |  
|-----|

|<Formed WP, without complementary FF's>  
|< that are removed through machining >

(3)!\*[ Demachine FF's\_of\_complementary\_forming ]\*  
v

-----  
| Form: First Process |  
| (inc.: aux. oper.) |  
| uni/multi-stage |  
|-----|

|<First-Process\_Formed WP>

(3.3)!\*[ DeForm Complementary Forming processes ]\*  
| ( i. e: DeIronning, DeSizing, DeExpanding, ... )

|<(n-1)-th process\_Formed WP>

(3.3)!\*[ DeForm\_Intermediate\_Forming ]\*  
| / (n-1)-th process\*/  
| ( s. g: DeReDrawing, DeSpinning\_Intermediate\_Pass )

-----  
| Form: n-th Process |  
| (inc.: aux. oper.) |  
| uni/multi-stage |  
|-----|

|<n-th-Process\_Formed WP>

(3.2)!\*[ DeForm\_Basic\_process ]\*  
| ( DeDraw, DeSpin, ... ).

-----  
| Form: Sup. Process |  
| (inc.: aux. oper.) |  
| uni/multi-stage |  
|-----|

|<Formed WP>

(4)!\*[ DeForm\_A\_Squeezing\_Process ]\*  
v

-----  
| Locate/ Purchase |  
| Raw-Material in |  
| in Stock. |  
|-----|

|<Std. RM shape.>

FINAL GOAL OF PROCESS-PLANNING

fig. 2.8.1 - One Concept of A Multy-Expert Process\_Planning System.

## 2.9 ABOUT APPLICABILITY.

The applicability and generality of a research do not always go together. Engineering research is often directed to a class of problems which is quite clearly bounded. Extensive elaboration is required to convert a general methodology to a working application. None the less, results can be extended beyond the original field of application. It is hoped, this system will not be an exception to this rule.

### Usage & Extensions

Some of the potential uses of the DDFE within a plant are:

- A more thorough check of producibility of designs at the design stage.
- Make the process\_planning function more efficient.
- Discover new manufacturing possibilities.

The motivation to automate the machining processes, - to save tedious, recurring, manual work, optimize parameters, utilize more extensive knowledge and expertise, etc., - is valid with all other metal working processes.

A successful realization of the process\_planning design for the primary parts ( i. e. : not including fabrications ) opens the route to "DeFabrication" - decomposition of the fabricated part into primary workpieces, and "DeAssembly" - decomposition of the assembled part into primary workpieces or fabrications.

### 3. GENERATING FEASIBLE PROCESS OUTLINES FOR FORMING PROCESSES.

#### 3.1 STRUCTURE OF FEASIBILITY

Feasibility is intuitively perceived as technical workability, but automating the process\_outline generation requires a formal, complete definition of feasibility. Since technical feasibility alone, would have led to infinite number of process outlines, the need to tie feasibility to "soundness" or "cost viability" is obvious.

The following brief discussion of feasibility relates process\_planning parameters with it.

A process\_outline is feasible, IF  
Every operation of the sequence of processes is feasible,  
The sequence, as a whole, is feasible.

For any intermediate process, including the concluding one:

An operation is feasible IF  
The material (raw\_workpiece ) entering the process conforms to the required prototype Initial Workpiece,  
The changes of features of the finished ( outcoming ) shape conform to the prototype changes of the process,  
The changes of features of the finished ( outcoming ) shape are executable,  
The appropriate and sufficient equipment exists ( main machine and auxiliary equipment ).

The first operation should yield to an additional condition:

First ( Initial ) operation is feasible IF  
The material (raw\_workpiece ) entering the process conforms to the required prototype Initial Workpiece,  
The the changes of shape features and mechanical features of the finished ( outcoming ) shape conform to the prototype changes of the process,  
The the changes of features of the finished (outcoming) shape are executable,  
The appropriate and sufficient equipment exist ( main machine and auxiliary equipment ),  
Raw\_Material of the required type, form, condition, & quantity is available.

The sequence as a whole is feasible IF  
Management requirement are satisfied ( quantity, lead-time ),  
The over-all cost of the sequence is in the order of or less than some reference process, usually:  
machining. ( if that process is capable of producing the desired properties ).

The cost criterion, which is the only non-technical criterion, is introduced in order to block against generating technologically-feasible but logically absurd process\_outlines. E.g.: If turning on a lathe can obtain the required surface finish honing should not be considered.

The complementary process to verifying feasibility is: parametrization. In parametrization the process is assumed to be feasible and the detailed instructions to carry it out are filled.

### 3.2 THE ROLE OF FEASIBILITY TESTS IN AUTOMATIC PROCESS PLANNING

The first stage in the process of selecting the "best" ( or: "optimal" ) process\_outline is generating the candidate process\_outlines. These candidates should withstand the feasibility tests, as defined in the section above.

Whereas in machining feasibility is a clear-cut measure, being satisfied usually allows implementation of the process, - in metal forming, the actual process parameters are matter of additional experiments and technological development. The successful accomplishment of the process is not guaranteed. Hence, feasibility check in metal forming corresponds to process-selection in machining.

In machining, frequent feasibility tests are:

- i. If the tool can reach the machined area,
- ii. If the process is capable of producing the type of feature,
- iii. If machine sizes suffice,
- iv. If the machine precision withstands requirements

In forming processes, feasibility would test, in addition:

- i. If conditions for start\_of\_flow of material are satisfied,
- ii. If the flow of material can be accomplished without a defect / failure been encountered.

Because of the complicity of the analytical analysis, and the variety of affecting factors, and their relative weight, for each process, these feasibility checks assume distinguished importance.

### 3.3 ASSUMPTIONS ABOUT FEASIBILITY OF A FORMING PROCESS

The following assumptions lay the foundations for the validity of the feasibility discussion in the deep\_drawing processes:

- I. If some parameter is within the required range, it is assumed, the concrete selection will be successfully accomplished, during the actual parametrization.  
e.g.: if control of friction can bring us to values in the range of: [ 0.04, 0.08 ], and the actual feasibility required friction of 0.05, it is assumed that the exact value will be obtained.
- II. Die and punch can be designed successfully to produce the required shape.
- III. Feasibility can be decomposed. E.g.: feasibility of drawing a tapered wall can be separated from the checking the tapered shell radii.
- IV. Feasibility check can follow the extreme boundaries pattern. I.e.: for each feature examined either a lower or an upper bound can be set.

### 3.4 FEASIBILITY REQUIREMENTS OF EQUIPMENT and RAW MATERIAL.

#### Equipment Feasibility

As defined above availability of the appropriate equipment is a condition to carry out the operation, while availability of raw material conditions the gross process\_outline. Equipment is appropriate IF

- i. It is of the right class,
- ii. Its structural features of the main machine satisfy requirements, ( i.e: control type, number of strokes in a press, type of actuation, ... ).
- iii. Its capacity contains the requirements, ( e.g.: tonnage of the first stroke greater than the required blank holding force, ... )
- iv. Its sizes facilitate the work ( e.g.: bed size allows the part clamping, ... ).

In addition, each machine has a characteristic cost and other managerial characteristics ( restricted here to : average wait time ), that participate in the overall feasibility check of the process.

The DDFE check feasibility by the following rules:

The structural- requirements group consists of

- i. the control type ( NC, manual, ... ),
- ii. the actuation ( mechanical, hydraulic, pneumatic, ... ),
- iii. group of special\_machine oriented features ( e.g.: for a press: # of slides ).

Here, the feasibility goes by a scheme of discrete matching. Upper bound specification does not fit. A general rule may specify:

IF mechanical\_actuation is required  
THEN hydraulic\_actuation will do too.

OR:

IF a two-slide press is required  
THEN either a two-slide press or a three-slide press fits.

The scheme determines the applicable matching range.

The capacity group contains all the quantitative features of the main machine and the auxiliary equipment.

For a press, the capacity requirements include such features as:

- tonnage in the i-th stroke,
- length of the cushion,
- maximal speed of each stroke,
- accuracy ( defined as deviation of punch under max. force ),

The features of the machine that facilitate the execution of the process, should be either "greater\_equal\_than" ( most of the features ), or "less\_equal\_than" ( accuracy ) the computed requirements.

### Raw Material Feasibility

Raw material is appropriate IF

It is of the required material,

It is of the right form ( sheet, tube, rod, ... ),

- fiber direction dictates different features.

It is in the right mechanical condition,

Each raw material unit in stock can issue at least one workpiece,

There are enough units to provide for all the required quantity.

The mechanical condition property is of special complicity. If there is no exact matching with the exactly required condition for the forming operation, a heat-treatment should be considered. The heat-treatment for raw materials is a complicate operation, in practice, and in extracting the knowledge for its feasibility. Among the underscored check-ups for this preparatory operation:

- i. Check for the appropriate, available, proven, heat-treatment process.
- ii. Check for the appropriate heat-treatment furnace ( appropriate in terms of: sizes, temperature, temperature distribution, temperature tolerance, cooling facilities, ... ),
- iii. Check for the appropriate atmosphere ( air, neutral-gas, vacuum, ... )

in order to simplify the first stage of the DDFE it is assumed that only exact matching of the raw material will satisfy the requirement.

### 3.5 A GENERALIZED ALGORITHM FOR GENERATING COMPOSITE PROCESS OUTLINES.

The generalized algorithm for generating feasible process\_outlines follows a general, recursive, approach of "test and rectify". Proceeding, backwards from the finished required workpiece, unto a recognized\_in\_stock raw\_material and initial basic process, an initial process\_outline is generated. The complete sequence of the process\_outline is tested for feasibility, forwards, - from the first process, and if a feasibility fault is found, the process\_outline is rectified. The recursive test\_and\_rectify continues until no fault is found.

The logic for not following a generate\_an\_operation\_and\_test algorithm, is that since the intermediate twin-tuples of [ Initial\_shape, Process\_name ] are not definite an infinite number of process-outlines may be generated.

A schematic flow-diagram of this algorithm in fig. 3.5

Note: the flow chart below only demonstrates the main idea about: "test and rectify" - it does not present a ready\_to\_program algorithm.

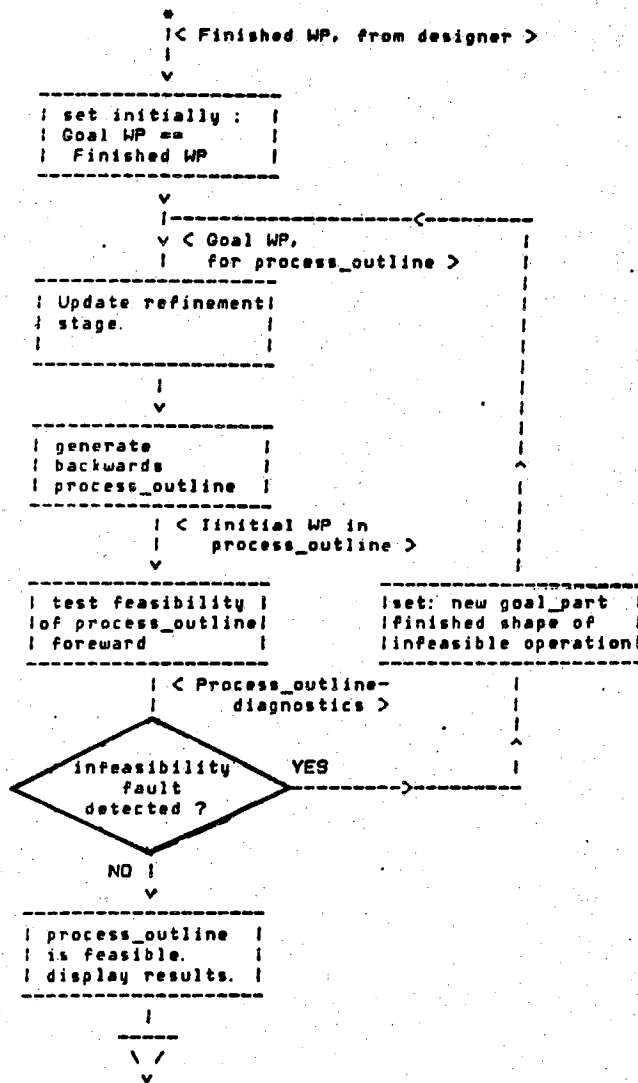


fig. 3.5 - Test\_and\_rectify flow-diagram



#### 4. DESIGNING FEASIBLE DEEP-DRAWING PROCESSES.

##### 4.1 A QUESTION TO A DEEP DRAWING EXPERT, and EXPECTED ANSWERS.

Since the Deep\_Drawing Expert is perceived as incorporated within the total automatic process\_planning system ( the generalized one, that includes non-chip producing processes ), consultations with him will take the form of:

" what is the combination of processes, in your field of expertise, that can produce the required end\_product "

"If the required end\_product is not within the range of expected end\_products ( either not in the form of the main shape of the process, or of deviating sizes ) produce an intermediate product, out of which the end\_product can be later extracted. "

This modification produces an interdisciplinary process\_outline, or: composite process\_outline.

The simplest composite process\_outlines are those composed of the main forming process and an auxiliary machining process. If the required end product can be extracted from the outcoming shell by machining only, the shell inscribes the end product - ( the end product is enclosed within the shell. )

Thus, for the composition of machining and deep\_drawing, in case some irregularity of the end\_shape is discovered, the deep\_drawing expert assumes he is allowed to produce an inscribing shape. In this case, his first operation will be: " inscribe the end product by a parametrized standard main shape of the process ". This enclosing is defined, symbolically, in appendix I-II, in the process independent predicate: "enclose". For the first stage development of the DDFE it is assumed the required parts are already beyond that preparatory stage. ( they fit the main\_shape of the process ).

If the required end product does conform to the standard prototype of the process, the expert is expected to come out with one of the following definite answers:

" The required end product is producible by the family of processes, and this is the feasible process\_outline ( the sequence of twin-tuples of initial shapes and processes ) "

OR:

" The required end product is not producible by the family of processes, and this is the first irregularity discovered during the check "

The DDFE leaves the task of modifying the design to the designer.

The expert infers the answer by utilizing his knowledge of the process flow\_of\_metal characteristics in the process, the general forming processes knowledge ( materials properties, fields of strains, ... ), and the plant resources ( raw\_material availability and equipment properties ).

Note, that the feasibility check is not an interactive session. It is a basic concept of automatic process\_planning that the inference of the process will be fully automatic and not interactive. Interactiveness requires some expertise in the manufacturing field, and it is because of this inexpertise, that automatic process\_planning systems are sought.

#### 4.2 APPLYING TEST AND RECTIFY FOR GENERATING DEEP DRAWING PROCESS OUTLINES

The methodology of generating a deep-drawing process\_outline, which is implemented in the predicate: GENERATE\_AND\_TEST\_PROCESS\_OUTLINE, in appendix I-II, is hereby described:

The root-rule predicate - GENERATE\_AND\_TEST\_PROCESS\_OUTLINE - can be implemented ( start the feasibility check )

if the required part (actually, the inscribing envelope, out of which the part can be machined. ).

If no need for machining, identify the candidate processes required for producing each of the elements of the final shape, and assign priority to each of them. Output of this stage: a list of candidate processes, with priority of application for the current\_final\_shape.

Backwards, generate a process\_outline such that , each twin\_tuple in the sequence consists of process\_name and the initial shape for that process, back unto the first stage.

The first process is distinguished from the intermediate processes by the fills of the twin\_tuple-slots: a basic process ( a process performed out of a standard raw\_material in stock ), and a standard raw\_material in stock.

For the initial candidate process\_outline generated: start the "test and rectify " procedure.

( The " test\_and\_rectify " process is currently implemented only for the basic process ),

For the raw\_material selected test feasibility of initial process. ( the predicate : FEASIBLE\_INITIAL\_PROCESS in the program performs this test ).

Feasibility of an initial process is validated if :

1. there exists an appropriate and sufficient material in stock.
2. an appropriate machine in the shop is available
3. can the required shape be worked in the basic process without defects from the stock raw\_material as is ( no heat treatment or any machining except for cutting the blank ).

Shape tests include check if the die and punch radii, are greater than the required minimum, for the draw\_ratio is less than the max. allowable draw\_ratio for the shell material.

If a fault is detected rectify process:

Rectification root-rule for the main\_process, rectifies process\_outline and retests. The test is comprehensive for every modification.

- i. IF raw\_material fail: Check for next possible thicker size ( if thicker plates allowed ), until no more available raw\_materials in stock.
- ii. If machine fail: - do not rectify, - stop\_and\_explain.
- iii. IF shape fail: check redrawing.

Incorporating complementary processes, and ironning - see scheme in & 3.5. These parts are not yet implemented.

THE METHODOLOGY OF INFERRING DRAWING-PROCESSES FEASIBILITY.

A Drawing\_expert that gets the specifications of the part required, first modify the shape to the std. output of the family of processes. This adjustment includes:

1. Prepare the required part for drawing feasibility\_check.
  1. enclosing the finished part with a minimal rotational, axisymmetric envelope.
  2. check for minimal fillet radii and mark them.
  3. modify marked radii ( if involves change of end\_shape - with designer ).
  4. Re-enclosing the axisymmetric envelope: increase wall thicknesses of segments of the envelope, so that the fillet radii of a segment will be connected by straight continuous lines, subject to certain rules of thinning.
  5. Check significance of mechanical properties: is a heat\_treatment required after end of draw ? - if the body does not undergo ironing,
  6. add allowances for finish machining.
2. Identify candidate complementary processes.
3. In conformance with priority table : Infer the initial input\_workpiece ( raw\_workpiece ) for the complementary process.
4. Perform stages #2, & #3 until a main-process output\_workpiece reached.
5. Check wall\_thickness changes and decide upon ironing.
6. Perform redraws, if required because of shape contour.
7. Check feasibility of main process. Infer if redraws to produce simple cup is required.

### 4.3 INPUT.

#### 4.3.1 CAM REPRESENTATION OF THE WORKPIECE and THE ULTIMATE GOAL: RETRIEVAL FROM C. A. D. DATA BASE.

CAM representation is the description of the workpiece in terms of the process it is bound to be produced. hence, the CAM representation of the workpiece is not unique and not general. It is process dependent. For machining purposes, the process generator should deal with form features - or rather surface features. Whereas for forming operations, volumetric description of the workpiece is required ( sometimes, together with surface features ).

Since axisymmetric shapes and axisymmetric\_shape Producing processes are evaluated, a concatenation of volumetric shape elements will suffice as an appropriate CAM representation. In the following paragraph a CAM representation of a simple shell.

The ultimate goal of the automatic process\_planning is to generate the process\_outlines directly from the CAD data\_base. In such an ideal system, a feature recognizer module would extract the information for the CAM representation of the workpiece. Another module would interpret and combine the extracted features to a set of workpiece properties ( one of them may be the GT class of the shape ). Features extraction directly from the CAD data\_base have been thus far, only conceptually proved feasible.

#### 4.3.2 FORMAT OF INPUTTING A QUESTION.

As noted above, conceptually, the required finished product description, is taken directly from the CAD database. Practically: the CAM interpretation of the CAD description is inputted directly. Then, assume: preliminary preparatory stage has been accomplished. - Main\_shape recognized / modified to conform with the requirements of the process.

In this case : the part was inscribed within a "minimal" rotational, axisymmetric, with bottom, envelope.

The part is represented as a four-tuple predicate in the following frame:

part (Part\_name, Part\_material, Part\_shape, Part\_requirements)

The part\_shape slot is nested within the part frame, being a frame in itself.

The part\_shape frame is composed of a list of elements, each of them is a frame. The element frame is:

- [
- Element\_name,
- Element\_type,
- Wall\_thickness,
- Inside\_diameter,
- Appropriate\_parameters  
    (- determined by 'element\_type' )
- Recess\_radius, ( Fillet-radius )
- ].

The corresponding frame for each parameter type determined by the parameter\_type slot of the element.

Frame for part requirements:

[ Required\_quantity, allowed Lead\_Time ]

The following shell-shape of fig. 4.3.2.1 is represented in the description in illustration 4.3.2.2 .

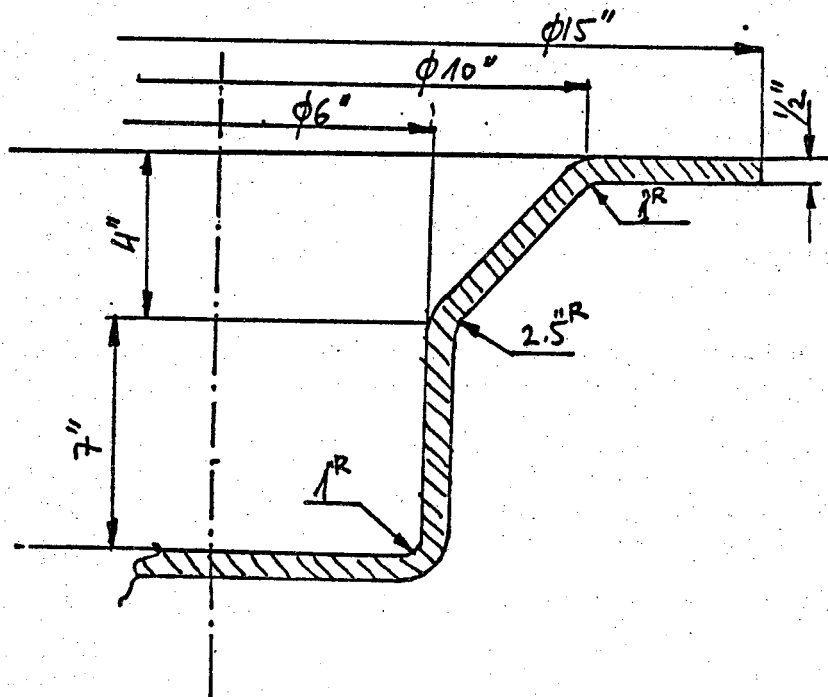


fig. 4.3.2.1 - shell shape.

```
part( part_3,  
      [[ st, 4130 ], [ 36, rc ]],  
      [  
        [ f, h, 1/2, 10, 15, 1 ],  
        [ a, a1, 1/2, 6, [10, 4], 2.5 ],  
        [ w, v, 1/2, 6, 7, 1 ],  
        [ b, h, 1/2, 0, 6, 0 ]  
      ],  
      [ 100, 21 ]  
    ).
```

illustration 4.3.2.2 - shell of fig. 4.3.2.1 - coded.

## 5. THE KNOWLEDGE OF THE DRAWING-PROCESSES EXPERT and ITS REPRESENTATION

### 5.1 GENERAL.

As noted above, a complete solution to the deep\_drawing problem is not available yet, only segments of the resultant behaviour of the deformed part in the process, are known. These practical segments fit the expert\_like approach synthesis: different models apply to different sets of conditions and contexts.

It has been found that this real life appearance of knowledge best fits a production system formulation. Hence, the general forms of knowledge representation in the DDFE are either production rules or facts. The rules are structured in the form of "Horn Clauses", and the facts in relational database. The Horn\_clause has one left hand predicate, which stands for the antecedent of the

"IF precedent THEN antecedent" form.

Thus formally, allowing only one consequent to any combination of conditionals. The one\_result form does not limit the actual result, only its formal representation, because any left\_hand predicate can be the antecedent of any set of conjunctions / disjunctions of right hand predicates. Both recursion and graph structure of rules stem from the fact that any right\_hand predicate may bear in itself a composite combination of precedents ( including itself ).

Since in many cases a rather natural description of knowledge may be a semantic network, an hierarchical structure of rules corresponds to this relationship. In this case the initial rule, out of which the tree branches down, is called: root-rule.

There are several layers of knowledge, according to the level of complicity and generality of the rules. The knowledge in each layer is a set of rules.

The layers, in descending order represented below:

1. Rectify a process\_outline.
2. Generate a process\_outline Define (generally) and infer feasibility of any intermediate process.
3. Define ( generally ) and infer feasibility of the basic ( initial ) process.
4. Define process capabilities ( any process ) to produce its parametrized std. shape, from a std., parametrized, raw\_workpiece.



5. Define process capabilities ( any process ) to produce its parametrized std. shape, in terms of the resources sought.
6. Define composite computations ( related to search the data\_base and manipulate the retrieved data ).
7. Define primitive matching of the data\_base : expand the PROLOG built-in capabilities to data\_base search.
8. Define low\_level computations.

## 5.2 ALGORITHMIC, PROCESS-INDEPENDENT KNOWLEDGE

The algorithmic, process-independent knowledge, or the generalized knowledge of the expert, is actually the high intelligence of the DDFE. It encompasses the following areas:

- i. General conditions for feasibility.
- ii. Generation of a next process
- iii. Generation of a process\_outline ( " test\_and\_rectify " ).

This high level knowledge is represented in the form of production rules with some common structural elements. The predominant common structural elements are:

- i. The call to the stop\_and\_explain function in cases of encountering a fault.
- ii. The definition of the feasibility conclusion.
- iii. The structured interpretation of the part and final shape information.

## 5.3 PROCESS CAPABILITIES KNOWLEDGE. - EXCERPTS

### 5.3.1 Contents of Process Capabilities

The detailed process\_capability knowledge for the deep\_drawing - main\_process - . The excerpts extracted in this paragraph are intended to exemplify the type of the practical engineering knowledge, which is constitutes the process capabilities files. In the appendix, the rules are either self explanatory or a preceding comment helps clarify them.

The process capabilities knowledge is the process-dependent knowledge. Here at least one variable of the predicate is instantiated. Most commonly, to the name of the process. The rule here is of the following general form:

```
IF
  ( process_conditions, initial_shape_properties )

THEN
  ( Required_Final_shape_feasible_within_the_following_range ).
```

Or, in Horn\_clause form:

```
predicate_name( Part, Initial_shape, Process, Final_shape ) :-
  combination_of(
```

{ sets of process conditions in terms of the initial\_shape }  
).

The three main areas of process capabilities rules are:

- i. Raw\_material and initial shape workable in the process.
- ii. Exact and Minimal machine requirements
- iii. Elements of the produced shape and their relationship to the initial shape ( of the particular material ).

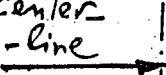



Some of the rules formulated in this paragraph are simplified for the cause of demonstration. ( putting them in the coded rule form is much more efficient ).

### 5.3.2 Assigning a workpiece to a process: verifying practical boundaries.

#### Description of Shape

As shown in & 4.3 shape is a structured concatenation of rotational rings, starting from the rim ( flange ). Each of the elements filled in the slot, is an frame in itself. The element frame, which is a six-tuple predicate, has, as its second variable, "type of ring shape" slot. The possible values of this slot, and their meaning, is following:

Range of values for the ring parameters:

|                                                                                   | <u>Type</u> | <u>Code</u> | <u>parameter</u> |
|-----------------------------------------------------------------------------------|-------------|-------------|------------------|
|  | -           | - h         | inside_diameter  |
|                                                                                   |             | - v         | -                |
|                                                                                   | /           | - o1        | + ( angle )      |
|                                                                                   | \           | - o2        | - ( angle )      |
|  | - r1        | - r1        | + radius         |
|                                                                                   | - r2        | - r2        | - radius         |
|  | - e1        | - e1        | + radius         |
|  | - e2        | - e2        | - radius         |
| )                                                                                 | - b1        | - b1        | + radius         |
| (                                                                                 | - b2        | - b2        | - radius         |
| U                                                                                 | - u         | - u         | + radius         |

5.3.3 Preliminary Constraints On Processed Shapes.

Sometimes, it is useful to employ global boundary limits, before going into detailed tests. By this principle, some rough features of the part are defined and checked through different predicates, in the process\_capabilities data\_base.

The approach by which the feasibility is checked through series of refinements, is adopted in other rule\_based process\_planning systems ( [ Descotte ], [ davies] ). To demonstrate the type of the feasibility constraints some excerpts for several deep\_drawing processes are presented.

The domain of deep\_drawing processes holds whence:

$$\frac{ID}{WT} > 10 ;$$

The following set some boundaries due to range of machines to work the drawings out. These are machine-dependent, factory-dependent constraints. The main machines that determine the following limits are:

Deep drawing.

- i. Presses sizes. ( largest opening & table sizes ).
- ii. Heat-Treatment furnaces ( highest possible temperature, furnace height and diameter ),
- iii. Lathes to turn the discs for the initial drawing

$$50 > OD > 1 ;$$

$$50 > H ;$$

Hydroforming

The hydroforming process produces a nominally uniform wall thickness shapes. The undesirable changes in wall thicknesses are of secondary order ( comparable to conventional deep drawing processes ). Control of that change, requires, currently, an engineering development, of a sophisticated, numerically controlled, process.

The process domain contains:

In addition to the Deep\_Drawing Constraints:

$$WT < \frac{3}{16} - \text{for Aluminium 2024, 7075}$$

$$WT < \frac{1}{16} - \text{for Stainless Steel .}$$

5.3.4 Shape Producing Capabilities.

( change in mechanical properties not presented here ).

Drawing, Main Process

Raw material: Circular Sheet / Plate.

In this process one draw is performed. The general constraints on shape produced by the Main\_process only, are:

$$D_i \geq D_j \text{ for any } j > i.$$

$$\frac{D_i}{\text{recess}_R_i} < 25$$

Deep\_Drawing\_Ratio for the first "cupping" operation is within the boundaries of: ( 25% .. 55% ). The highest values obtained by the maximum drawability alloys: stainless steel and copper alloys, while the lowest for refractory metals ( like columbium based and tungsten ). The limitation of one draw, limits the practical scope

of products, significantly.

Aside for the class of stretching processes, where normal anisotropy prevails, no pre\_planned strain\_hardening is considered.

From the technological point of view, the drawn part undergoes different type of process ( the resultant stresses is different, the dominant strain, ... ), depending upon its structural features. The features that determine the process behaviour are:

- i. relative depth of shell ( rough classification to: shallow - deep ),
- ii. relative wall\_thickness ( rough classification : thin -thick ),
- iii. main shape ( rough classification: tapered, straight\_walled, spherical, - the dominant shape determines the class ),
- iv. flanginess ( wide flanged cups, no\_flange cups, ... ),
- v. stretchability, ( rough classification would differentiate between shells that undergo stretching to those who do not )

A division of the main\_process, based upon a those technological classification of shapes is employed in the process\_capability rules. This classification turns to a different approximate model of computing allowed ( feasible ) drawing.

The class representation in the program takes a qwin-tuple predicate, excluding the depth\_of\_shell class, which is treated as the resultant variable, dependent upon the previous four.

Some typical factors in the classification ( not necessarily the full classification criterion as in the "classify\_X" rules ) are elaborated below:

The following rules pertain to shallow cups only:

Shallow drawings :  $H < \frac{4}{5} * ID$

Cupped : Bottom shape type: Main\_shape: horizontal.

flanged : flange : exists.

width\_of\_flange > 3 \* thickness\_of\_blank.

( an estimate to thickness of blank:

Bottom\_thickness )

not\_flanged: flange does not exist.

negation of conditions for flange existence.

Stretched cups : Bottom shape type: Main\_shape: r1

nominal desired, WT reduction:  $\frac{OD_{RM}^2 + h^2}{OD_{RM}^2}$

Shallow flanged cup: flange : exists.

width\_of\_flange  $\approx$  thickness\_of\_blank.

strain\_hardening:

assuming an empirical equation of the form:

$$\sigma = Y + e^n$$

corresponds to an engineering strain of:

$$\frac{3}{2} \ln \frac{OD_{RM}^2 + h^2}{OD_{RM}^2}$$

- assuming R values  $\left( \frac{\epsilon_{thickness}}{\epsilon_{length}} \right)$

in the order of : - 2;

( see R-values, in terminology ).

Schematic relationship between the limiting draw ratio, stock thickness and punch diameter for carbon steel illustrated in fig. 5.3.2.1 ( from: [ Hobbs ]. )

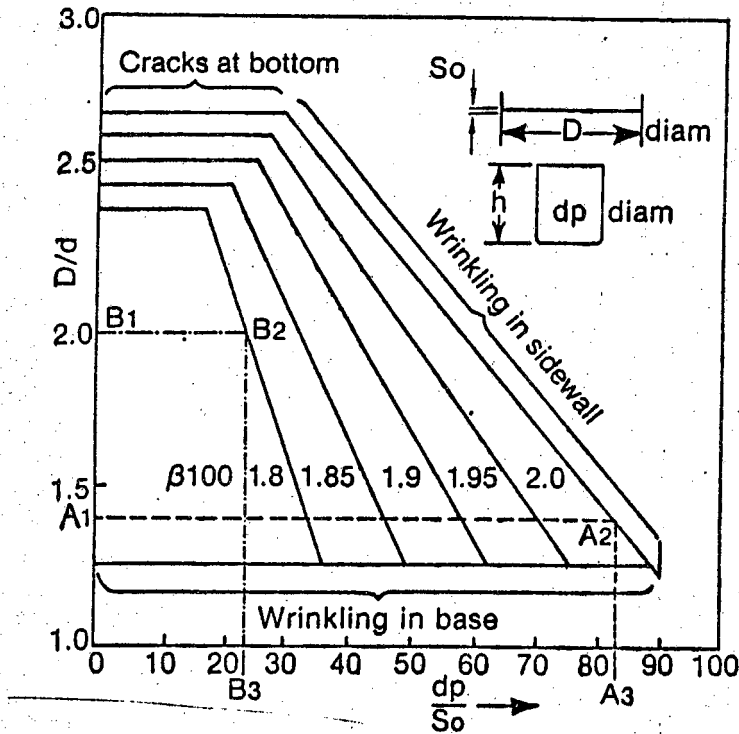


fig. 5.3.2.1 - limit draw\_ratio as a function of: WT, R, for steel.

aspects of differnt cupping of the main process

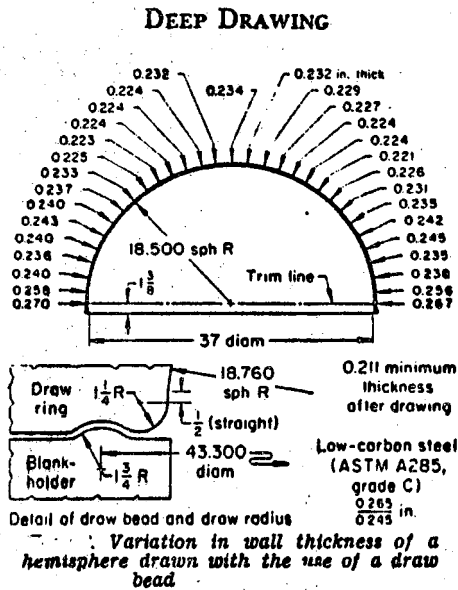


fig. 5.3.2.2  
different WT in drawn hemisphere  
( from: [ Lyman ] )

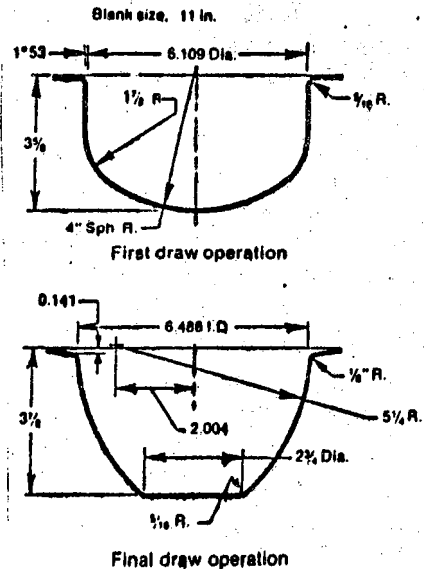


fig. 5.3.2.3  
two-stage taper drawing.  
( from: [ Lyman ] )



5.3.5 Shapes of complementary processes.

Redrawing :

Initial Shape

- i. Straight walled cylindrical segment.
- ii. Max. ratio of Deep\_Drawing\_Ratios:  $i^{th}$  redraw :  $i+1^{nd}$  redraw  
=  $1:\frac{2}{3}$

Product of Process

- i. A limit of 8 redraws is assumed, including intermediate heat-treatments.
- ii. The same shapes as of Deep\_Drawing - Main\_process, except for the limitations on resultant Deep\_Drawing\_Ratios and heights.

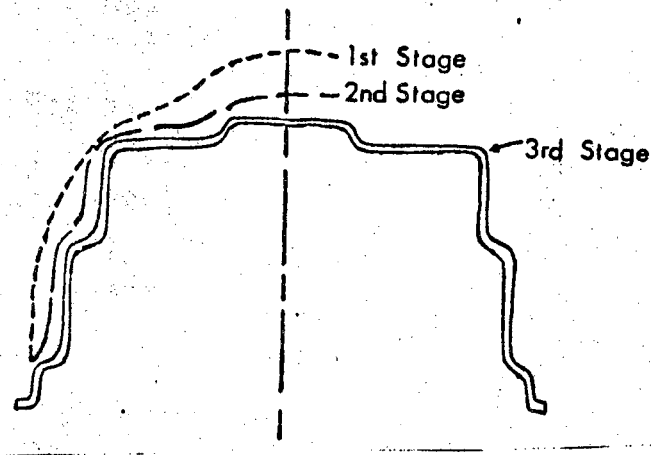


fig. 5.3.5.1  
typical redraws for a stepped cup  
( from: [ Hobbs ] ).

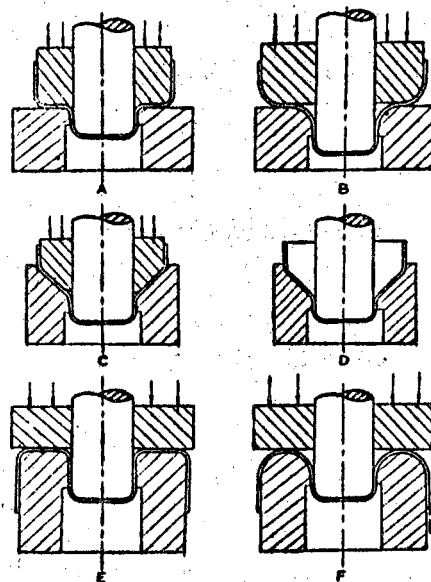


fig. 5.3.5.2  
diagramatic redraws:  
( from: [ Johnson ] ).

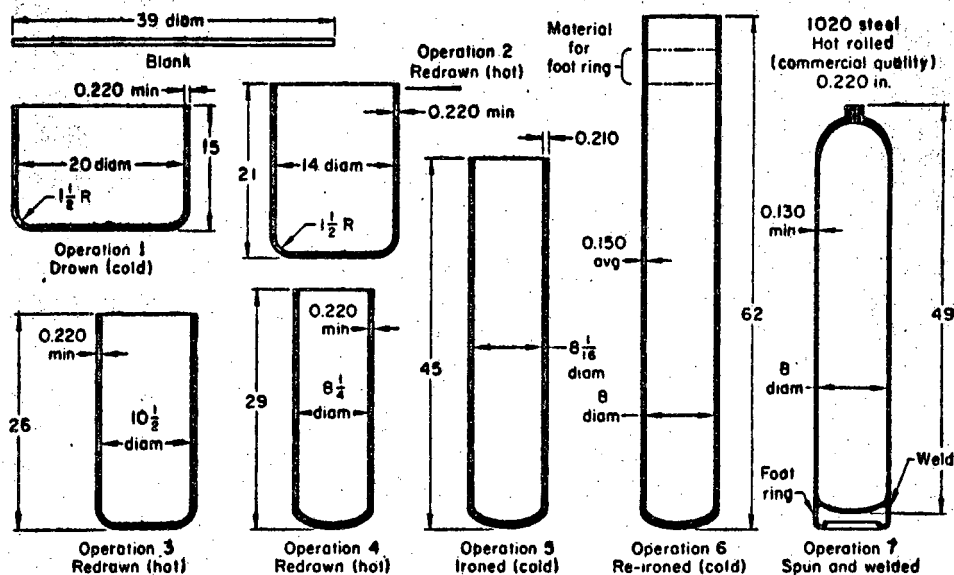
Ironing

Initial Shape

- i. straight cylindrical portion
- ii. rest of shape features conform to deep\_drawing - Main\_process with / without redraws.
- iii.  $D_{initial\_cylinder} > D_{ironed\_cylinder}$  by  $\approx 3\% - 5\%$ .

Product of Process

- i. thinned wall thickness of ironed segment.
- ii. uniform wall thickness.
- iii. can be used for correcting undesired thickening / wrinkles in wall.



|                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Presses:</b><br/>                 Draw .....1500-ton double-action hydraulic<br/>                 First redraw 500-ton double-action hydraulic<br/>                 Other redraws .....500-ton single-action hydraulic</p> <p><b>Lubricants:</b><br/>                 Draw .....Sulfonated oil<br/>                 Fourth and fifth redraws(a) .....Dry soap</p> | <p><b>Die materials:</b><br/>                 Hot operations .....Graphitic tungsten tool steel(b)<br/>                 Cold operations .....D2 tool steel<br/>                 Die hardness .....Rockwell C 60<br/>                 Tool life, pieces per grind(c) ..... 1000<br/>                 Lot size, pieces ..... 1000<br/>                 Annual production, pieces ..... 1000</p> <p>(a) Including ironing operations, which needed additional lubrication. (b) 1.50% C, 0.40% Mn, 0.65% Si, 2.80% W. (c) Tools were reconditioned after annual run.</p> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Fig. 5.3.5.3 - some typical uses of ironing and redrawing ( from: [ Lyman ] ).

Some typical shapes of necking, expanding and sizing.

( from: [ Lyman ]. )

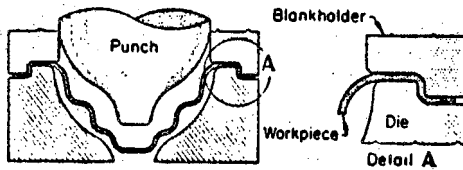


fig. 5.3.5.4  
reduction - necking

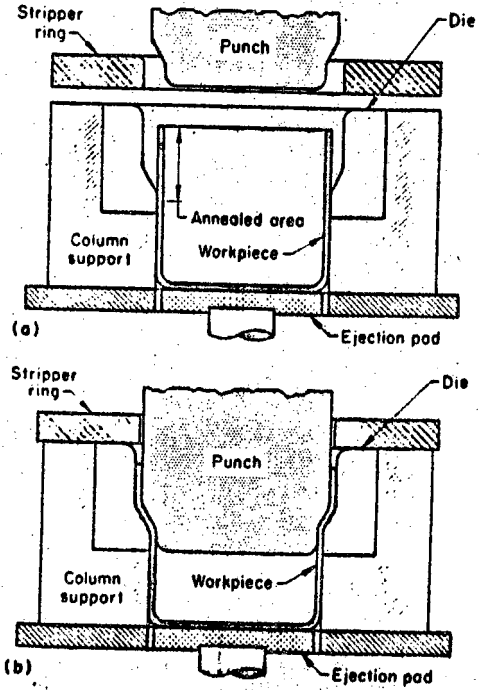


fig. 5.3.5.5  
expanding - regular

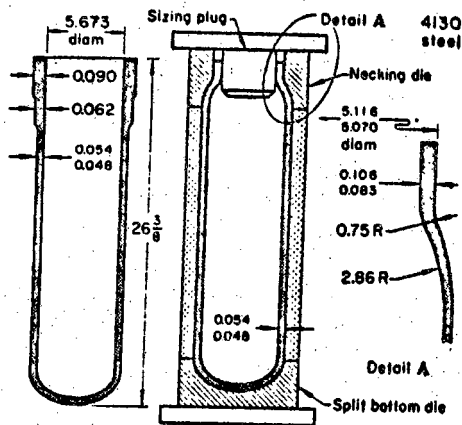


fig. 5.3.5.6 - sizing

#### 5.4 RESOURCE KNOWLEDGE

The resource knowledge includes the properties of the raw materials in stock and the equipment capabilities. This knowledge, which is later termed as : dynamic data\_base ( because it represents a current state in the plant ) is represented as facts. A PROLOG fact is of the form:

prediacte\_name( Instantiated variables ).

These facts are represented and interpreted as frames, with all slots filled ( all the variables are instantiated ).

#### 5.5 GENERAL TECHNOLOGICAL KNOWLEDGE

The general technological knowledge pertains to technological facts which are not process related ( or process\_dependent ). Such facts are the formability properties, the machinability ratings, per particular material, and empirical results about workpiece behaviour during the forming processes.

Such designated facts are:

max. draw ratio,

min. allowed die radius,

heat\_treatment requirements between stages ( passes ).

The general technological knowledge is represented as structured frames, and grouped in

static data\_base. It is named static, although it is bound to be constantly updated, because it is current\_state-independent.

#### 5.6 EXTRACTING UPDATING and DEVELOPING THE DRAWING PROCESSES EXPERT KNOWLEDGE.

The knowledge of any forming processes expert is extracted from a combination of: textbooks, research articles, summarized shop tables, handbooks for material properties, maintenance documents ( for machine capabilities ) and the plant's resource data\_base.

Extracting the process-capability knowledge is the core and the foundation of a forming-processes expert system. It is conceived to constitute the next stage after accomplishing the process modeling phase. Process modeling of forming processes, is quite a new research area ( see references. See NAMRC Conferences for the last 7 years, [ Chen ], [ Thomas ]. The referenced researches try to characterize the flow of the metal, in the deformation zone, as a function of the material and process parameters.

Putting the process capabilities knowledge into a ( any ) formal representation is yet another difficult, innovative, and

challenging task. The experience of formulating and building the first module of the DDFE system, suggests that this task may be successfully accomplished only if the persons building the system master the domain expertise and the knowledge engineering art.

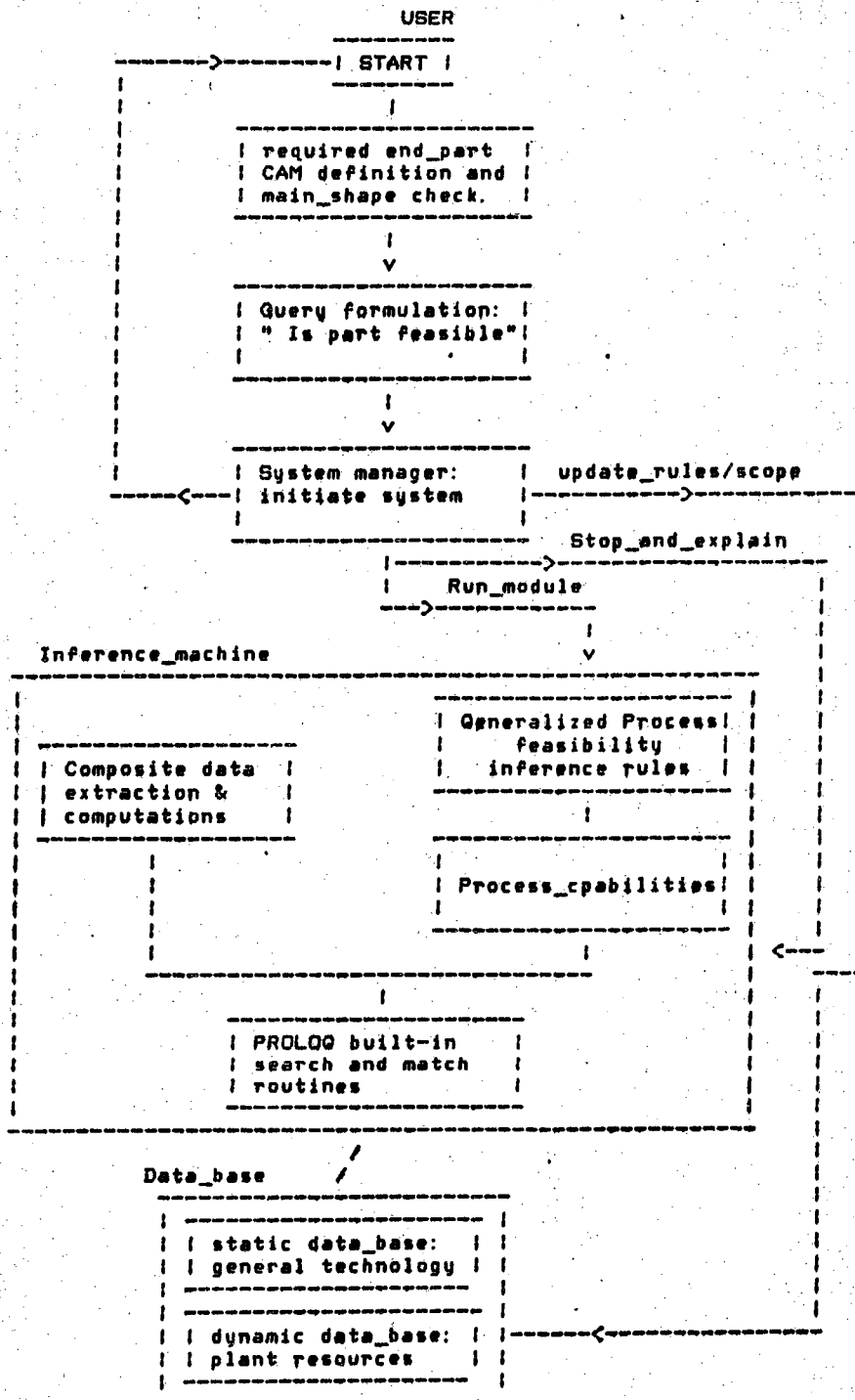
The process-capabilities knowledge of the DDFE is extracted from:

- i. Textbooks, - the analytical analysis. ( references [ Lyman ], [ Hobbs ], [ Wick ] ), represent a real knowledge source for this portion ). The textbooks furnish the approximate-idealized models for computing draw\_ratios forces, and outgoing strain-hardening.
- ii. Research papers, especially the ones elaborating deep-drawability, and formability limits. ( for example see references [ Koistinen ], [ Niemeier ] ).
- iii. Shop tables, are a kind of unique knowledge every forming engineer collects throughout his industrial experience. It refers to such topics as: lubrication rules, blank holding devices, and die-punch spacing to prevent wrinkling and puckering. Because of the enormous work done in composing such curves, the specific conditions, in which the experiments were taken, are of utmost importance.
  - tables
- iv. Materials handbooks are very important day-to-day tools for extracting the main properties affecting formability, such as:
  - yield and ultimate tensile strength
  - ductility characteristics: reduction of area, elongation,
  - characteristic anisotropy during stretching,
  - strain hardening curve,
  - change of properties with temperature.
- v. Maintainance documents - usually, in the plant the maintainance department has the most accurate records about machine capabilities. Current capabilities and the theoretical ones ( machine performance as specified by the manufacturer ). Plant data-base is the source of the state of raw\_materials in stock, functioning facilities ( main and auxiliary machines ) and shop managerial practices, such as cost of machine hour and typical lead-time.

The underlying principle in the DDFE, and other Expert Systems, is the independence of data of the program. Here, the independence assumes an additional aspect: the lower-level rules can be modified, let alone the static and dynamic data base. A user-friendly mode of updating both the data\_base and the lower-level rules is initiated by calling the " up\_date\_X " files.  
( Thus far only tentative action is specified as a comment ).

6. THE DDFE SYSTEM.

6.1 SYSTEM STRUCTURE.



## 6.2 SYSTEM MANAGER.

Since the DDFE is not intended to be an interactive system, the managing modules are designated to guide the user;

- how to initiate the system,
- how to put in his question, and
- how understand the system evaluation.

The system manager groups the following modules:

- i. System communicator
- ii. Knowledge Updating guide
- iii. Explainer
- iv. System parametrizer

The system communicator is organized in files:

" start ", " explain ", " a ", " message ".

Following the typing in of "start" the user is guided through the screen, to proceed the session. The message and the consultation are demonstrated in appendix I.

The system knowledge updating guide sets the scope of updating the knowledge. It is intended to explain the user the allowed range of modifying the knowledge, lead him how to append, update and modify knowledge, in each of the modifyable files. This module is symbolically represented in the "update\_.. " files. ( not complete yet ).

The explainer is the program stop\_and\_explain, which is invoked whenever a feasibility fault is discovered. It is in charge of :

- stopping the feasibility evaluation session,
- getting out of the PROLOG mode,
- explaining the user the design fault found.

A demonstration of the job it is doing - in appendix I.

The system parametrizer is intended to set the parameters of the cost-feasibility function. The parametrizer specifies the standard comparative process in comparison to which cost feasibility is evaluated, and the order\_of\_the\_cost parameter. A process will be considered feasible if its estimated cost is below X times the cost of the standard process. - that X is the order\_of\_the\_cost parameter. ( at this stage : machining from a solid block is taken to be the measure of cost-feasibility ),

### 6.3 THE INFERENCE MACHINE and HIERARCHICAL RULE STRUCTURE.

The implementation of the reasoning knowledge representation, as elaborated in § 5, is in a hierarchical rule structure. Implementing the rule structure, the general inference machine and the partially instantiated rules for the process capabilities, in Horn\_clauses, facilitates the PROLOG application. Conceptually the inference machine is built of the following modules :

1. The generalized knowledge: structured rules.
2. Formal representation of the capabilities knowledge base: instantiated rules.
3. Matching: PROLOG matching and backtracking, and DDFE composite matching.

As noted above, all the rules, through all the inference machine, are in clausal form, thus perpetuating, naturally, a top-down rule structure. This structure, is actually a net-structure, and not a tree structure, because "lower level" clauses are embedded in different clauses which "stem" from different roots. The predicates employed in the different branches / layers of rules are not confined to general search procedures, but include functional clauses too. ( e.g. : the classify predicates,

DDFE adopts the PROLOG depth-first backtracking search strategy to explore alternative branches of the search space. The order in which clauses are written determines the order in which they are tested. Thus, a clause of the form:

X :- A, B, C.

will attempt to solve X by testing A first then B and lastly C. Once a goal, say C, fails, PROLOG tries to resatisfy B. Once B is resatisfied, the search to resatisfy C resumes, from the beginning. This mode of search goes on until all the data\_base is searched or, other means ( "cuts", "cut-fail"s, ... ) clauses can not be resatisfied. Further discussion of these features can be found in [ Clocksin ], [ Deyi ].



#### 6.4 THE DATA BASE: RELATIONAL DATA BASES and FRAMES.

As noted above, both the dynamic and static data\_bases are frame based. The conceptual frame leads to an implementation of the representation in a "flexible relational data\_base". The conceptual flexibility is of a restrained degree of freedom in the slot filling. It allows different types of variables and composite structures to be filled in every slot. This embarks on the PROLOG property of equally treating variables and nestedlists, which actually assume every possible network structure.

For example: The variable "Material" in the predicate: "rm" ( raw\_material ):

```
where :   rm( [ _!Material ] ),
```

Material can be instantiated to :

```
steel;  
[ steel, 4130 ];  
[ [steel, 4130], [annealed] ];  
[ [steel,[ sae 4130]], [ 36, rc ]]; ...
```

#### 6.5 WORKING WITH THE DDFE.

The DDFE is currently capable of evaluating the main question it is designed for: " Can the particular end\_workpiece be manufactured in the deep drawing process ? ". Nonetheless, other answers are extractible, provided that an appropriate predicate for them is defined. These expansions are in the process of development.

After initiating the system the user is guided how to read\_in the knowledge files, and prepare the part to be evaluated. While the part is evaluated, the intermediate important conclusions are recorded, or if a design fault is discovered, the user is immediately informed about it, and the evaluation is terminated. The procedure of working with the system is described in appendix I .

#### 6.6 FORMULATING THE PROBLEM FOR THE DDFE.

The user should write the description of the part, its producibility he wants to verify, in a designated file: " db\_part ". The frame for the part specification is explained in the file, while it is invoked. Sample parts, tested and stored in the file "part", and their decoded meaning, in appendix I .

## 7. TESTING AND EVALUATING THE DDFE.

### 7.1 CURRENT SCOPE OF PRODUCTS AND PROCESSES.

Thus far only the main\_process module and partial knowledge of the redrawing and ironing have been programmed. Running the DDFE with sample parts, largely demonstrated a dependable capability. Some results of the runs to evaluate the system appear in appendix I.

The following conclusions can be drawn with regard to its current capability and future forecasted reliability:

- i. The range of feasible change of thickness in an intermediate element has not yet been fully defined.
- ii. Four of the six main defects are not yet fully defined.

### 7.2 THE COMPUTATIONAL EFFORT.

The DDFE is written as a subset of the PROLOG C-interpretter, in the UNIX system. Because the internal data\_base representation in PROLOG is not user controlled, some assumptions about randomness and uniform distribution for entity search, have been introduced.

Reading ( "Consulting", in the PROLOG terminology ) the current files takes about 25 sec., one third of them for the data\_base files. This means that, were a real knowledge base to be consulted, it would have taken tens of minutes. ( introducing only 100 items of stock would add half the current consultation time ). The time performance here is estimated to increase proportionally to the data\_base size:

-  $O(n)$ , where  $n$  is the number of data\_items in the data\_base.

As for the program execution time:

The two modes of search, in the DDFE differ significantly in performance. While there are  $\approx 100$  rules, searched sequentially, the real performance problem lies with the data\_base search. Since the depth-first search of PROLOG [ Clocksin ] is adopted, the computational effort is largely dependant upon the size of the data\_base. Changing the order of sequence of the materials in the data\_base ( in the "DB\_STOCK" file ), alone, increased the execution time from 1.7 sec. to 3.3 sec. for a successful consultation. The depth-first tree search of the data\_base here, yields an average performance measure of:

$O( ( \log_2 \text{ number\_of\_tables} ) \times \# \text{\_of\_data\_items\_in\_a\_table} )^2 )$ .

Hence, from the computer resources point of view, this system can not fit a real plant needs. In order to do it an interface to a more efficient DBMS should be introduced. An attempt to solve the PROLOG problem by this means, in [ Deyi ].

## 8. ABOUT RELATED EXPERT SYSTEMS.

AI applications in CAM, and especially the utilization of expert systems to process\_planning has grasped the imagination and effort of the recent CAM research. But thus far, very few systems passed the conceptual definition stage and none is available beyond the academic environment of its designer.

Two systems have been reported in papers. GARY [ Descotte ] is the oldest and the most known one. GARY uses a rule based system ( about fifty rules reported to have been formulated ), to generate, in a series of refinements, a process\_plan to machine parts. The report elaborates only about prismatic parts. The reasoning ability of the system, is quite limited, because it follows a rigid structure of preparatory metal-removal and finish-machining. When a contradiction among assertions ( or pieces of advice ) is detected, the system discards the previously applied piece of advice, and updates the current solution. Gary is implemented in MACLISP on the HB-68, under the MULTICS system.

GARY, unlike DDFE, attaches a weigh to each piece of advice, a measure which enables it to discriminate between assertions when a contradiction is detected. Machines are described by their properties, and each property ( e.g.: type\_of\_machine\_is\_chuck\_lathe ) is examined in the program, according to the set of rules, which checks for it.

GARY uses a series of refinements to produce the final program. The DDFE, which is concerned with verifying feasibility, before going to the detailed process\_plan, resembles GARY in that it generates a rough process\_outline first, and employs a recursive procedure to refine it.

The second, recently reported process\_planning system, EXCAP [ Davies ], developed at UMIST, England. EXCAP is designed to generate process\_plan to machine rotational parts ( axy-symmetric ). It uses a fuzzy logic rule structure, to introduce the element of uncertainty, in the form of:

IF < to a certain extent > condition  
THEN < to some degree > action.

From the paper, it appears, as if the system is partly interactive, and not all the decisions are automatic.

In comparison to the DDFE, its predominant enhancement is the fuzzy logic. But, it should be noted, the DDFE is designated to verify feasibility, and feasibility should not be fuzzy, - rendering the uncertainty measure, here, redundant.

## 9. EXTENSIONS AND FURTHER DEVELOPMENT.

As mentioned throughout the report, much of the effort, put in thus far in the system, is directed towards completing and expanding the system. With the current generalized inference modules the following process can be readily incorporated, once the process\_capability knowledge is formalized:

- i. Redrawing,
- ii. Sizing : Flanging,  
          Contouring,
- iii. Reducing: Necking,  
          Nosing,
- iv. Expanding: Regular,  
          Bulging,
- v. Embossing.

Accomplishing a more rigorous definition, of the main\_process, redrawing and ironing, requires some, relatively small changes in the instantiated ( to process name ) root-rule for process feasibility. The appropriate way to take care of such expansions and future modifications, is by:

defining a supervisory rule for inferring one\_process feasibility, that would search for all the instantiated feasibility predicates of the examined process.

Next envisioned expanding stage is to embed a data\_base management system with the DDFE, so that rules will be searched by the prolog interpreter, directly, while, facts through interfacing the DBMS.

Another step in enhancing the utilization of the system is to facilitate additional queries. The most likely way to do this is by defining a set of predicates, each of them can be instantiated independently.

10. BIBLIOGRAPHY

Note: whenever there is more than 2 authors only the first one is mentioned here, together with the addendum: "et al".

1. Andersen, B.S. : A numerical Study of The Deep-Drawing process, in: Pittman, J.F.T. et al, ed. : Numerical Methods In Industrial Forming Processes, Pineridge press, U.K. 1982.
2. Chen, C.C. (ed.) : Experimental Verification of Process Models, Proceedings of Symposium, Cincinnati, Ohio, Sept. 1981, American Society of Metals, 1982.
3. Clocksin, W.F., Mellish, C.S. : Programming In PROLOG, Springer-Verlag, 1981.
4. Davies, B.J., Darbyshire, I.L. : The Use Of Expert Systems In Process-Planning, in: Annals of CIRP, Vol. 33/1/1984.
5. Descotte, Y., Latombe, J.C. : GARI : A problem solver that plans how to machine mechanical parts, IJCAI, Aug. 7-th, 1981, Vancouver, Canada.
6. Deyi, L. : A PROLOG Data\_base System, John Wiley & Sons, 1984.
7. Hobbs, R.M, Duncan, J.L. : Press Forming, Advanced technology Course # C21L4, American Society for Metals, 1979.
8. Hecker, S.S., et al (ed.) : Formability Analysis, Modeling, and Experimentation, Proceedings, Oct. 1977, Chicago, Ill, American Society of Metals, 1978.
9. Ghosh, K.A. : " Plastic Flow properties In Relation To Localized Necking In Sheets, in: Koistinen, D.P., Wang, N.M. ( ed. ) : Mechanics Of Sheet Metal Forming, Plenum Press, 1978.
10. [ Johnson1], Johnson, W., Mellor, P.B. : Engineering Plasticity, Joun Wiley & Sons, 1976.
11. [ Johnson2], Jounson, W., Mamalis, A.G. : A Survey of Some Physical Defects Arising in Metal Working Processes, in: ed. Tobias, S. A. : 17-th Intl. Machine Tool Design and Research Conference, Birmingham, Sept. 1976.
12. Koistinen, D.P., Wang, N.M. ( ed. ) : Mechanics Of Sheet Metal Forming, Plenum Press, 1978.
13. Lee, D. : Computer Aided Control of Sheet Metal Forming Processes, in: Journal of Metals, Nov. 1982.
14. Lyman, T. ( ed. ) : Metals Handbook, Vol. 4: Forming, American Society for Metals, 1969.

15. Niemeier, B.A., et al (ed.): Formability Topics - Metallic Materials, symposium, May, 1977, Toronto, Canada, American Society for Testing Materials, 1977.
16. Niwa, K., et al: An Experimental Comparison of Knowledge Representation Schemes, in: The AI Magazine, Summer 1984.
17. Pittman, J.F.T. et al, ed.: Numerical Methods In Industrial Forming Processes, Pineridge press, U.K. 1982.
18. Slater, R.A.C.: Engineering plasticity, John Wiley & Sons, 1977.
19. Stefic, M., et al: The Organization Of Expert Systems, A Tutorial, in: Artificial Intelligence 18(1982), 135-173.
20. Semenov, D.I., Bersenev, V.A.: Semiotic Models In Geometric Design Automation, in: Blake, P.: ed.: Advanced Manufacturing Technology, North Holland, IFIP, 1980.
21. Thomas, J.F., Dadras, P.: Modeling Of Sheet Forming Processes - An Overview, in: Chen, C.C. (ed.): Experimental Verification of Process Models, Proceedings of Symposium, Cincinnati, Ohio, Sept. 1981, American Society of Metals, 1982.
22. Wanheim, T., et al: Physical Modelling of Metal Forming Processes, in: Journal of Applied Metal Working, American Society For Metals, Vol. 1 No. 3-5, 1980.
23. Wick, C., Benedict, T.J., Veillieux, R.F.: Tool and Manufacturing Engineers Handbook, Vol. 2: Forming, Society of Manufacturing Engineers, 1984.
24. Weil, R., et al: Survey of Computer Aided Process Planning Systems, in CIRP Annals, 1981.
25. Woo, D.M.: On The Complete Solution of The Deep-Drawing Problem, in: Intl. Journal of Mech. Sciences, Vol. 10, 1968.

13. Appendix I - SAMPLE DDFE RUNS and RESULTS.

Sample Parts:

The parts were coded into file db\_part.

```
-----  
/*          File name:  DB_PART          */  
  
/* File contains the coded information about the enclosing part */  
/* It is assumed that, by a certain procedure the finished_part */  
/* has been enclosed by an inscribing envelope throughout      */  
/* preparatory stages.                                          */  
/* If that stage has not yet been accomplished, or; the part is */  
/* already in the form of a drawable workpiece -                */  
/* - proceed directly from this file.                            */  
  
/*          FRAME for part          */  
  
/* part(Part_name, Part_material, Part_shape, Part_requirements) */  
/*   FRAME for Part_shape: a list of axisymmetric elements:     */  
/*     FRAME for element:                                       */  
/*       [                                                       */  
/*         - Element_name,                                       */  
/*         - Element_type,                                       */  
/*         - Wall_thickness,                                     */  
/*         - Inside_diameter,                                   */  
/*         - Appropriate_parameters                             */  
/*           (- determined by 'element_type' )                  */  
/*         - Recess_radius, ( Fillet-radius )                   */  
/*       ]                                                       */  
/* The corresponding FRAME for each parameter_type given in text */  
/*   FRAME for Part_requirements:                                */  
/*     [ Required_quantity, allowed Lead_Time ]                 */  
  
part( part_1,  
      [ [ st, 4130 ], annealed ],  
      [  
        [ f, h, 1/2, 8, 11, 4.5 ],  
        [ w, v, 1/2, 8, 4, 2.1 ],  
        [ b, r1, 1/2, 8, [3, 4], 0 ]  
      ],  
      [ 400, 21 ]  
    ).  
  
part( part_2,  
      [ [ st, 4130 ], [ 36, rc ] ],  
      [  
        [ f, h, 1/2, 10, 15, 1 ],  
        [ w, v, 1/2, 7, 5, 2.1 ],  
        [ b, r1, 1/2, 7, [15, 4], 0 ]  
      ],  
    ),
```

[ 400, 21 ]  
).

```
part( part_3,  
      [[ st, 4130 ], [ 36, rc ]],  
      [  
        [ f, h, 1/2, 10, 15, 1 ],  
        [ b, a1, 1/2, 6, [10, 4], 2.5 ],  
        [ w, v, 1/2, 6, 7, 1 ],  
        [ f, h, 1/2, 10, 12, 0 ]  
      ],  
      [ 100, 21 ]  
).
```

```
/* end_of_file : "db_part" */
```

---



SAMPLE RUNS:

Initiating The System:

user types: "start";

system returns:

WELCOME TO THE CONSULTATION WITH  
DDFE

- Deep-Drawing Feasibility Expert System -

The session is carried out within a PROLOG interpreter  
You will be automatically carried into the PROLOG mode  
The terminal will return with a ' ! ? ' sign,  
Then, please type in : '[zz].'  
This command calls for the all the data\_base files to be consulted  
After each file is consulted, you will get a message on the screen  
The consultation takes about 24 seconds.  
In the end of the consultation you will get a message:  
' consultation finished successfully.'  
Then, please print: [ < file\_name\_of\_your\_part > ].  
GOOD LUCK WITH THE SESSION.

----- user types: [zz].

----- system returns:

CProlog version 1.3

! ?- [zz].

db\_draw\_ratio reconsulted 4728 bytes 1.25 sec.  
db\_draw\_sphere reconsulted 340 bytes 0.150001 sec.  
db\_drawing\_force reconsulted 604 bytes 0.266668 sec.  
db\_equipment reconsulted 1852 bytes 0.5 sec.  
db\_forming\_properties reconsulted 744 bytes 0.283334 sec.  
db\_materials reconsulted 1404 bytes 0.383335 sec.  
db\_part reconsulted 1476 bytes 0.366667 sec.  
db\_process\_class reconsulted 1036 bytes 0.35 sec.  
db\_radius reconsulted 1228 bytes 0.400002 sec.  
db\_shape\_cap reconsulted 3140 bytes 0.7 sec.  
db\_stock reconsulted 1732 bytes 0.316667 sec.  
comp\_blank reconsulted 2416 bytes 0.700001 sec.  
comp\_draw\_ratio reconsulted 1300 bytes 0.416667 sec.  
comp\_force reconsulted 1396 bytes 0.450003 sec.  
enclose reconsulted 0 bytes 0.116667 sec.  
explain reconsulted 532 bytes 0.166668 sec.  
extract\_shape\_data reconsulted 8800 bytes 2.95 sec.  
infer\_sequence reconsulted 4692 bytes 1.71667 sec.  
prelim\_process\_match reconsulted 2788 bytes 0.850003 sec.  
print\_data reconsulted 3412 bytes 0.900005 sec.  
priority reconsulted 1588 bytes 0.666667 sec.  
process\_shape\_capability reconsulted 8480 bytes 3.11667 sec.

process\_machine\_required reconsulted 1736 bytes 0.666672 sec.  
select\_control reconsulted 0 bytes 0.0333415 sec.  
rule reconsulted 3484 bytes 1.46667 sec.  
rm\_search reconsulted 3092 bytes 1.2 sec.  
procedures reconsulted 4520 bytes 1.06667 sec.  
update\_dynamic\_db reconsulted 76 bytes 0.116669 sec.  
update\_static\_db reconsulted 0 bytes 0.100004 sec.

Consultation completed successfully

Your part is represented in file x1. type in "[x1]".

In order to get out of the PROLOG mode, type "[a]".

message reconsulted 0 bytes 0.133335 sec.

zz consulted 66596 bytes 22.4833 sec.

yes  
! ?- [x1].

----- user types: [x1].

----- system returns:

CONSULTATION SESSION COMPLETED SUCCESSFULLY

please type in "[a]." and read results in file "kovez"

x1 consulted 0 bytes 1.56667 sec.

yes  
! ?-

----- user types: [a].

----- system returns:

Prompt\_sign  
and the user looks into the specified file.  
-----

Advise About Part-1:

Desired Finished Part Specifications  
=====

Part name is - part\_1  
Required Material - [[st,4130],annealed]  
Finished Part Inscribed Within The Following Envelope  
-----

The structured form of the workpiece is

f, h, 1/2, 8, 11, 4.5,  
w, v, 1/2, 8, 4, 2.1,  
b, rl, 1/2, 8, [3,4], 0,

Quantity and Due date required - 400 units, 21 weeks from now.

The EXPERT has learned your required part.

Start of generating a feasible process for the part  
=====

The Candidate processes are -  
-----

[deep\_drawing,[main\_process,draw]]  
[deep\_drawing,redrawing]

Current Sequence of Processes And Shapes Tested -  
-----

Process name - [deep\_drawing,[main\_process,draw]]  
Raw workpiece tested is of the shape of - BLANK  
Its sizes are :

Wall thickness - 1/2  
Blank diameter - 16.4424

-- Next Process --

Process name - inspection  
Desired Shape At end Of Process  
f, h, 1/2, 8, 11, 4.5,  
w, v, 1/2, 8, 4, 2.1,  
b, rl, 1/2, 8, [3,4], 0,

Feasibility Check For Generated Sequence of Processes Starts-  
-----

Important intermediate findings will be reported by:  
/ FOR INFORMATION ONLY /

/ FOR INFORMATION ONLY /

The blank computed to produce the required shape is :

Blank wall tickness = 1/2

Blank diameter = 16.4424

Form of raw\_material accepted in this process is: flat

OPERATIONAL INFORMATION

-----

applicable material found

The raw material found in stock is

-----

material name - rm\_3

Material form - flat

Material sizes -

Wall Thickness -1/2

Width -36

Length -96

Quantity - 50

Material Type - [st.4130] and condition - annealed

/ FOR INFORMATION ONLY /

Computed machine requirements:

-----

Machine build requirements are:

Any legal value

hydraulic

Any legal value

Machine capacity requirements are:

3619.11

Any legal value

8.39999

Any legal value

Any legal value

Any legal value

16.4424

12

OPERATIONAL INFORMATION

-----

appropriate machine found

The appropriate machine is press6

Machine type - press

Structural features -

Control - nc

Actuation - hydraulic

Number of slides 2

The maximum tonnage is - [4000,6000]

Accuracy, in mm. - 0.2

The stroke length and cushion are - [300,400], [80,10]  
Strokes\_per\_minute and Slide\_velocity are - 15, [800,600]  
Bed\_opening and Upright\_opening are - 40, 30  
Cost\_per\_hour ( Dollars ), and Characteristic\_wait\_time are - 100,

/ FOR INFORMATION ONLY /

List\_of\_draw\_ratios :

[Cup\_diameter/Blank\_diameter, 2.0553]

[( Blank\_diameter / Sphere\_Opening ) / Sphere\_factor , 1.54147]

/ FOR INFORMATION ONLY /

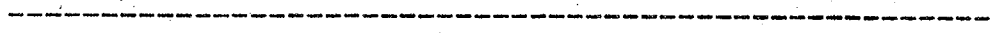
Elements of shape drawn are classified as:

[spherical, thin, not\_flanged, no\_stretch]

The following shape features have been evaluated:

- Fillet\_radius at flange
- Fillet\_radius at bottom
- Draw ratios of current shape

all have been found workable !



Advise About Part-2:

Desired Finished Part Specifications

=====  
Part name is - part\_2  
Required Material - [[st,4130],[36,rc]]  
Finished Part Inscribed Within The Following Envelope  
=====

The structured form of the workpiece is

f, h, 1/2, 10, 15, 1,  
w, v, 1/2, 7, 5, 2.1,  
b, r1, 1/2, 7, [15,4], 0,

Quantity and Due date required - 400 units, 21 weeks from now.

The EXPERT has learned your required part.

Start of generating a feasible process for the part

=====  
The Candidate processes are -

-----  
[deep\_drawing,[main\_process,draw]]  
[deep\_drawing,redrawing]

Current Sequence of Processes And Shapes Tested -

-----  
Process name - [deep\_drawing,[main\_process,draw]]  
Raw workpiece tested is of the shape of - BLANK  
Its sizes are :  
Wall thickness - 1/2  
Blank diameter - 27.2506

-- Next Process --  
Process name - inspection  
Desired Shape At end Of Process  
f, h, 1/2, 10, 15, 1,  
w, v, 1/2, 7, 5, 2.1,  
b, r1, 1/2, 7, [15,4], 0,

Feasibility Check For Generated Sequence of Processes Starts-

-----  
Important intermediate findings will be reported by:  
/ FOR INFORMATION ONLY /

/ FOR INFORMATION ONLY /

The blank computed to produce the required shape is :

Blank wall thickness = 1/2

Blank diameter = 27.2506

Form of raw\_material accepted in this process is: flat

\*\*\*\*\*

Violation of feasibility condition:

Discovered in process: locate appropriate raw material in stock

The initial shape of this process is : exactly\_blank\_wt

The conditions that could not be satisfied: Too few sheets

Discovered during: Computation of quantity

the EXPERT advises you to recheck your specifications.

Preceding features and tests are in file: "kovez".

\*\*\*\*\*

---

---

**Chapter 5**

**An Expert System For Machine Selection of FMS**

*S. Lan*



## AN EXPERT SYSTEM FOR MACHINE SELECTION OF FMS

Sheree Lan

### 1. INTRODUCTION

#### 1.1 Background

The concept of flexible manufacturing systems (FMS) is very appealing to low volume and mid-volume manufacturing. In these types of manufacturing, both the productivity and flexibility of the manufacturing system are of major concern. An FMS consists of a number of numerical control (NC) machines, a material handling system, and a computer control system. The machining system (NC machines) and the material handling system are automated and controlled by the computer system. By computer-integration of NC machines and the material handling system, FMS can easily adapt to different jobs and still achieve high productivity.

The design of FMS is a very complicate task. It takes time from months to years. The design process of FMS follows the configuration-analysis-modification loop iteratively until good designs are found [BARA79]. Most sources available in literature of FMS design deal with problems of using simulational and analytical techniques for analyzing some known FMS designs. Very few, if not none, try to identify or formalize approaches needed to design initial configurations of FMS. The initial configurations of FMS have always been built up by some rules of thumb [KLAH83] and by designers' experience [IT082]. If the knowledge of FMS configuration can be extracted from human designers and

put into the form of an expert system, then the FMS design task can be sped up significantly. These configurations then can be simulated or analyzed, and modified configurations obtained.

The most important component of an FMS is its machining system. Usually, this accounts for the largest expenditure of an FMS [LEWI73]. Also, the configuration of the machining system has the largest influence on the capability of the system. It is felt that the configuration of machining system is the first step in FMS design. Once the machining system is configured, the performance of different material handling systems can be modeled and analyzed rather easily.

## 1.2 Statement of problem

This report is to describe the design of an expert system for machine selection of FMS. Machines under consideration are just generic machines. The purpose of the expert system is to decide the types and quantities of machines required to fulfill the manufacturing of some known products. Only prismatic workpieces (workpieces in box shape) are considered. The information of product design and process planning is assumed available.

## 2. SYSTEM ORGANIZATION

The designed expert system consists of four main components: rule base, operation base, machine base, and stack base. The inference program looks for the adequate solution of machine combinations based on these four components.

### 2.1 Rule Base

The rules are expressed in the (IF.. Then.. ) form. Rules are put in a file and read by the program, which will convert the rules, and store them in a linked list. The rules consist of "Simple fact" and defined "Function". "Simple facts" are less than 10 characters long words, represent the situation of certain stage or flag. "Functions" are eight characters long words specified by the knowledge engineer. They are coded in the program, users can only use defined functions while change rules. There are some attribute names for functions. Each attribute name represents one attribute in operation base. The program will recognize the attributes of rules and access the attributes information from operation base.

### 2.2 Operation base

For this system, the information of work contents is collected and put in a file which is accessible to the system. All required processes of all products are put in a table, attributed by their features, as shown in Figure 1. According to the defined format and order, users can input the operation data to

the file, the system can recognize these data and put in a linked list of records.

### 2.3 Machine base

The machine configuration is changed when one possible rule is applied. The machine base links all operations in each group. The rules can easily apply to adequate machines and operations, change the operation base and machine base simultaneously.

### 2.4 Stack base

The stack base is a record file, which keeps the inferencing procedure information. Any change in operation base, rule base, and machine base will be recorded, formalized in stack record and write to filestack. The stack base takes the advantage of various type of files in PASCAL language. The run time working space can be saved and backtracing is easier to perform.

### 2.5 Inference program

This is the program which checks the rules, searches for the one that the condition part is satisfied, and trigs the consequence action. The search of a satisfactory rule is done by the order priority strategy. To improve the efficiency of searching, rules are divided into two groups, and the configuration procedure is divided into three stages. The first group of rules are for the 'grouping' of operations. The second group of rules manage to delete under-loaded machines and specialize well-loaded machines, so that a good combination is obtained.

## 2.6 Control strategy

The strategy selected is order priority, mainly because it is easy to implement and also it does not cause any confliction while using the system.

## 2.7 Information Management

The rule base is represented in IF..THEN form. The simple facts are 10 characters long words. At the beginning of each rule, the global contexts can be present to represent repetition of operation type, horse power type, and special operation type. They are 10 characters long words :

1. forallopty - for all operation types
2. forallhpty - for all horse power types
3. forallspop - for all special operation

The defined functions are 8 characters long followed by a '('.  
The defined functions are listed as follows :

1. machload( - machine load
2. machexis( - machine exists
3. transfer( - transfer load from machine one to machine two
4. attrload( - attribute load in a machine
5. attraxis( - attribute exists in a machine

6. attrtran( - transfer load for an attribute from one machine to another
7. defoptyp( - define operation type number
8. defhptyp( - define horse power type number
9. nofactex( - no such fact exists
10. delefact( - delete the fact

The attributes are represented as 8 characters long words followed by a ',' and their ranges. The attributes are listed as follows :

1. opertype - operation type, by a specific operation type
2. houpower - horse power, by lowerbound, to upper bound
3. spoperat - special operation, by a specific operation
4. wkspleng - workspace length, by range of length
5. wkspwide - workspace width, by range of wide
6. wksphigh - workspace height, by range of height
7. feedrate - feedrate, by range of feedrate
8. yrdemand - demand, by range of demand

### 3. KNOWLEDGE ACQUISITION

The knowledge required in selecting machines is the 'grouping' and 'combining' rules involved. Some of these rules are mentioned in journal papers and industrial magazines [KLAH83]. Some of them are the result of the computer simulation runs, or from analysis. At this moment, only rules of the first type are used. However, it is possible to interface simulation or analytical programs with the rule base. All these rules can be explained by economic analysis from different levels. For examples, some rules try to eliminate machines with insufficient loading, assign their work content to other machines. Some rules will reduce the flexibility imposed on the machine to improve the efficiency of the machine. The logic behind these rules is summarized in the following subsection.

#### 3.1 Logic of machine selection

Selecting machines for some workpieces can be viewed as an function of matching work contents and machine capability. If the intended products for an FMS is known, then the process planning system will generate the processes required to manufacture them. Each process is characterized by some features, which are the implicit form of work content. Some of the important features are machine power, work space, degree of motion, and type of operations.

Machine power required for operations can be calculated from process plans. Usually, higher powered machines are of higher

prices. It is expected to assign the work contents (operations) to appropriate machines, so that the operations can be performed under the best conditions (i. e. cutting speeds, feeds, etc.), and also the machine is not overpowered.

Work space is concerned with the working envelop required while machining a workpiece. A machine over-facilitated of workspace not only cost the initial investment but also decrease the efficiency.

By degree of motion capability, it means the type of positioning and cutting motion the machine is able to operate. While under capacitated machines cannot achieve the work required, over capacity in degree of motion of machines increases the initial cost.

Operation types include drilling, milling, boring, reaming, etc. One special type of operation is the multi-spindle operation. Machines specialized for multi-spindle operations are available, for examples, head indexers, head changers, and special drilling machines. Usually, machining centers can operate with small heads. For large-sized heads, head-changing machines are required.



## 4. EXPERIMENTAL RESULTS

### 4.1 Runs

Two illustrating examples have been run on this expert system and the results are analyzed. The sample run screen script file of Run 1 is included in Appendix. According to the program designing, the operation base can be added at some intermediate stage of inferencing procedure. The rule base can also be reset at anytime. Run 4 is a sample illustrating the operation base input at different stages. Run 3 is the comparing example of Run 4, but input operation base all at once.

In Run 1, there are 29 operations running. In first result combination, there is no machine horse power less than 10 of operation type 1 and 3 existing. If the user feel the cost of using machine horse power less than 10 is cheaper than using machine horse power between 10 and 20, the user can go back to the stage that applying transforming from horse power less than 10 to horse power between 10 and 20.

In Run 2, there are 21 operations running. These operations do not have many type 3 operations. There are a lot of operations of type d in type 1 operations, and type f in type 2 operations. The loads of these operations are comparatively less than Run 1 data.

In Run 3, they are the combination of Run 1 and Run 2. The operations are input together at the beginning. The special

machines are introduced more than in Run 1 and Run 2.

In Run 4, they are also combination of Run 1 and Run 2, but the operations are input at different time. The results are slightly different from Run 3.

#### 4.2 Evaluation

The running time used is varied by the given operation data. The average time used to perform a 30 operations system is about 35 cputime. The backtracing will be more time consuming, but comparing to human, it is still faster.

The user can change the rules i.e. total number of operation type, horse power ranges for machines, and the priority of rule order. If the user has different grouping methods, he can also use the defined functions and attributes to implement his own rules.

The system can use several different rules at run time. The rules change the context and apply different strategy as the user needed. Only one thing, the system does not check the ambiguous situation. The strategy is always rule order priority.

As we had shown above, the system can input operation at certain stages of inferencing procedure. Also, the system can keep the backtraced information in a file (filestack). When next time the system is called, the previous information is still there, and save computing time.

## 5. CONCLUSIONS

The rule base we used in sample runs is a general rule for configuring. The smaller operation type will be grouped to larger operation type if it has low load. The lower horse power operation will also be grouped to higher horse power machine if it has low load. So, initially, the larger operation type and higher horse power machine will be used as much as possible. The result is optimal but when concerning efficiency may not be optimal. Thus, the certainty factors should be introduced here.

Since the certainty factors are not used in this system, the on line users should have some knowledge of their needs. The system will search the first combination. The rule applied and stack information can be backtraced. If the user does not like some machine being used, he can ask the system to go back to the stack number right before the machine is generated, or before the load of that machine is increased by transferring other machine load to it.

REFERENCES

[BARA79]: BARASH, M. M., et al, Optimal Planning of Computerized Manufacturing Systems, NSF Grant No. APR 15256, A Process Report, School of Industrial Engineering, Purdue University, 1979.

[BARR81]: BARR, A., Feigenbaum, E. A., "The Handbook of Artificial Intelligence", vol 2, 1981, 1982.

[HAYE83]: HAYES-Roth, F., Waterman, D. A., Lenat, D. B., "Building Expert System", 1983.

[ITO 82]: ITO, Y., et al, "Description of Machining Function in FMS and its Analysis", Proc. of MTDR Con., 1982.

[KLAH83]: KLAHORST, H. T., "How to Plan Your FMS", Manufacturing Engineering, Sept. 1983.

[KOFF81]: KOFFMAN, E. B., "Problem Solving and Structured Programming in PASCAL", Addison-Wesley, 1981.

[LEWI73]: LEWIS, F. A., "Some Factors Affecting the Design of Production Systems in Batch Manufacture", Proc. of MTDR Con., 1973.

[McDEB1]: McDERMOTT, J., "R1 The Formative Year", AI Magazine, Summer, 1981.

Figure 1 : Data Organization Table

| Partid   | optype | operation | optime<br>min | workspace<br>l*w*h | HP<br>hp | tl1n<br>ft | feedrate<br>cuft/min | deman<br>#/yr |
|----------|--------|-----------|---------------|--------------------|----------|------------|----------------------|---------------|
| 00010001 | 1      | d         | 2.37          | 3*4.5*2            | 10       | 1.4        | 1.56                 | 2000          |
| 00010002 | 1      | r         | 1.65          | 4*5*1              | 15       | 2.0        | 3.44                 | 2000          |
| 00010010 | 2      | m         | 3.00          | 1*1*1              | 30       | 0.5        | 0.11                 | 2000          |
| 00020001 | 1      | t         | 2.50          | 1*1*1              | 15       | 1.1        | 1.22                 | 1200          |
| 00030001 | 1      | d         | 2.90          | 1*1*1              | 25       | 1.2        | 0.90                 | 900           |

Data specification :

operation type : 1 - 2 axis positioning  
2 - 2 axis contouring  
3 - 3 axis  
4 - 4 axis  
5 - 5 axis

operation : d - drilling  
m - milling (2 axis and milling)  
t - tapping  
r - reaming  
b - boring  
c - counterboring  
f - finish  
h - hole

operation time : (minutes / part)

work-space required : L \* W \* H (cu. in.)

horse power required : (hp)

tool length : (in.)

feed rate : (in. / revolution)

demand : ( year demand)

maching loading = total machine time / annual time per machine.

annual time per machine = 250 \* 24 \* 60 \* demand

RUN 1

- 250 -

Script started on Tue Dec 4 04:49:57 1984  
\* obj

251

This program will allow the users to

- 1) reset (change) "Inference Rule", its IF part and THEN part. Only those "Simple Fact" and defined "Complicated Fact" can be used as the reasons. The rules are in order priority.
- 2) input operation data.
- 3) run the inference procedure.

Please enter the option (1,2,3 or q) 1

What is the (simple fact) file name : filefact  
What is the rule file name : filerule

This program will allow the users to

- 1) reset (change) "Inference Rule", its IF part and THEN part. Only those "Simple Fact" and defined "Complicated Fact" can be used as the reasons. The rules are in order priority.
- 2) input operation data.
- 3) run the inference procedure.

Please enter the option (1,2,3 or q) 2

Input operation data in a file (y/n)? y  
What file ? lan

This program will allow the users to

- 1) reset (change) "Inference Rule", its IF part and THEN part. Only those "Simple Fact" and defined "Complicated Fact" can be used as the reasons. The rules are in order priority.
- 2) input operation data.
- 3) run the inference procedure.

Please enter the option (1,2,3 or q) 3

What stack number do you want to start with ?  
Please enter the stack number : 0

0.900

IF  
stage0  
nofactex(define)

THEN  
defoptyp(3)  
defhptyp(3)  
define

1.000

IF  
attrexis(opertype,1,0,0,0)

THEN  
attrtran(opertype,1,0,0,0,1,0,0,1.0)

2.000

IF  
attrexis(opertype,2,0,0,0)

THEN  
attrtran(opertype,2,0,0,0,2,0,0,1.0)

3.000

IF  
attrexis(opertype,3,0,0,0)

THEN  
attrtran(opertype,3,0,0,0,3,0,0,1.0)

4.000

IF  
attrexis(opertype,4,0,0,0)

THEN  
attrtran(opertype,4,0,0,0,3,0,0,1.0)

6.000

IF  
nofactex(stage1)  
machexis(-1,0,0)

THEN  
delefact(stage0)  
delefact(define)  
stage1

7.000

IF  
stage1  
machexis(-1,0,0)  
attrexis(houpower,0.0,10.0,-1,0,0)

THEN  
attrtran(houpower,0.0,10.0,-1,0,0,-1,1,0,1.0)

```
8.000
IF
stage1
attraxis(houpower,10.0,20.0,-1,0,0)

THEN
attrtran(houpower,10.0,20.0,-1,0,0,-1,2,0,1.0)

9.000
IF
stage1
attraxis(houpower,20.0,1000.0,-1,0,0)

THEN
attrtran(houpower,20.0,1000.0,-1,0,0,-1,3,0,1.0)

10.000
IF
stage1

THEN
delefact(stage1)
stage2

11.000
IF
stage2
machaxis(1,-1,0)
machload(1,-1,0,0.0,0.7)
machload(2,-1,0,0.3,100.0)

THEN
transfer(1,-1,0,2,-1,0,1.0)

25.000
IF
stage2
machaxis(3,1,0)
machload(3,1,0,0.0,0.7)
machload(4,1,0,0.0,0.3)
machload(3,2,0,0.3,100.0)

THEN
transfer(3,1,0,3,2,0,1.0)

35.000
IF
stage2
machload(-1,-1,0,1.8,100.0)
attrload(spooperat,*, -1,-1,0,1.8,100.0)

THEN
attrtran(spooperat,*, -1,-1,0,-1,-1,*,0.5)
```

The result arrangements of operations are :



stack number : 13  
rule applied : 35.000

35.000

stage2  
machload(-1,-1,0,1.8,100.0)  
attrload(spooperat,\*, -1,-1,0,1.8,100.0)

THEN  
attrtran(spooperat,\*, -1,-1,0,-1,-1,\*,0.5)

current existing facts :  
1 stage2

current arrangement of operations (by parttoolid) :

| partid      | mach | hp | special | load  |
|-------------|------|----|---------|-------|
| 1 00010001  | 1    | 2  | 0       | 0.728 |
| 2 00010002  | 1    | 3  | c       | 0.556 |
| 3 00010003  | 1    | 3  | 0       | 1.750 |
| 4 00010004  | 1    | 2  | 0       | 0.519 |
| 5 00010010  | 2    | 3  | 0       | 0.587 |
| 6 00010020  | 1    | 3  | 0       | 0.103 |
| 7 00020001  | 1    | 3  | 0       | 0.719 |
| 8 00020002  | 1    | 2  | 0       | 0.488 |
| 9 00020003  | 1    | 2  | t       | 0.381 |
| 10 00020004 | 1    | 2  | b       | 0.483 |
| 11 00020010 | 2    | 3  | 0       | 0.728 |
| 12 00020011 | 2    | 1  | 0       | 0.800 |
| 13 00020021 | 3    | 2  | 0       | 0.656 |
| 14 00020031 | 3    | 3  | 0       | 0.453 |
| 15 00020032 | 3    | 2  | 0       | 0.659 |
| 16 00030001 | 1    | 3  | 0       | 0.625 |
| 17 00030001 | 2    | 1  | 0       | 0.604 |
| 18 00030002 | 1    | 3  | c       | 0.438 |
| 19 00030003 | 1    | 2  | t       | 0.415 |
| 20 00030004 | 1    | 2  | b       | 0.463 |
| 21 00030010 | 2    | 1  | 0       | 0.458 |
| 22 00030011 | 2    | 3  | 0       | 0.208 |
| 23 00040001 | 1    | 3  | 0       | 0.229 |
| 24 00040002 | 1    | 3  | c       | 0.094 |
| 25 00040010 | 2    | 1  | 0       | 0.302 |
| 26 00050001 | 1    | 3  | 0       | 0.669 |
| 27 00050002 | 1    | 2  | 0       | 0.463 |
| 28 00050003 | 1    | 2  | t       | 0.151 |
| 29 00050020 | 3    | 3  | 0       | 0.204 |

current arrangement of operations (by machines) :

|                                             |        |       |
|---------------------------------------------|--------|-------|
| machine 1 2 0 include parttoolid : 00010001 | load : | 0.728 |
| machine 1 2 0 include parttoolid : 00010004 | load : | 0.519 |
| machine 1 2 0 include parttoolid : 00020002 | load : | 0.488 |
| machine 1 2 0 include parttoolid : 00050002 | load : | 0.463 |
| machine 1 2 0 total load :                  |        | 2.197 |
| machine 1 2 t include parttoolid : 00020003 | load : | 0.381 |
| machine 1 2 t include parttoolid : 00030003 | load : | 0.415 |
| machine 1 2 t include parttoolid : 00050003 | load : | 0.151 |
| machine 1 2 t total load :                  |        | 0.947 |

```

machine 1 2 b include parttoolid : 00020004      load :    0.483
machine 1 2 b include parttoolid : 00030004      load :    0.463
machine 1 2 b total load :                0.945

```

```

machine 1 3 0 include parttoolid : 00010003      load :    1.750
machine 1 3 0 include parttoolid : 00010020      load :    0.103
machine 1 3 0 include parttoolid : 00020001      load :    0.719
machine 1 3 0 include parttoolid : 00030001      load :    0.625
machine 1 3 0 include parttoolid : 00040001      load :    0.229
machine 1 3 0 include parttoolid : 00050001      load :    0.669
machine 1 3 0 total load :                4.095

```

```

machine 1 3 c include parttoolid : 00010002      load :    0.556
machine 1 3 c include parttoolid : 00030002      load :    0.438
machine 1 3 c include parttoolid : 00040002      load :    0.094
machine 1 3 c total load :                1.087

```

```

machine 2 1 0 include parttoolid : 00020011      load :    0.800
machine 2 1 0 include parttoolid : 00030001      load :    0.604
machine 2 1 0 include parttoolid : 00030010      load :    0.458
machine 2 1 0 include parttoolid : 00040010      load :    0.302
machine 2 1 0 total load :                2.165

```

```

machine 2 3 0 include parttoolid : 00010010      load :    0.587
machine 2 3 0 include parttoolid : 00020010      load :    0.728
machine 2 3 0 include parttoolid : 00030011      load :    0.208
machine 2 3 0 total load :                1.524

```

```

machine 3 2 0 include parttoolid : 00020021      load :    0.656
machine 3 2 0 include parttoolid : 00020032      load :    0.659
machine 3 2 0 total load :                1.316

```

```

machine 3 3 0 include parttoolid : 00020031      load :    0.453
machine 3 3 0 include parttoolid : 00050020      load :    0.204
machine 3 3 0 total load :                0.657

```

Do you want to see the intermediate steps (y/n)? y

```

stack number :    0
rule applied  :    0.000

```

```

current existing facts :
  1 stage0

```

current arrangement of operations (by parttoolid) :

| partid      | mach | hp | special | load  |
|-------------|------|----|---------|-------|
| 1 00010001  | 0    | 0  | 0       | 0.728 |
| 2 00010002  | 0    | 0  | 0       | 1.112 |
| 3 00010003  | 0    | 0  | 0       | 1.750 |
| 4 00010004  | 0    | 0  | 0       | 0.519 |
| 5 00010010  | 0    | 0  | 0       | 0.587 |
| 6 00010020  | 0    | 0  | 0       | 0.103 |
| 7 00020001  | 0    | 0  | 0       | 0.719 |
| 8 00020002  | 0    | 0  | 0       | 0.488 |
| 9 00020003  | 0    | 0  | 0       | 0.762 |
| 10 00020004 | 0    | 0  | 0       | 0.966 |
| 11 00020010 | 0    | 0  | 0       | 0.728 |

# RUN 2

stack number : 11  
rule applied : 35.000

35.000

stage2  
machload(-1,-1,0,1.8,100.0)  
attrload(spooperat,\*,-1,-1,0,1.8,100.0)

THEN  
attrtran(spooperat,\*,-1,-1,0,-1,-1,\* ,0.5)

current existing facts :  
1 stage2

### current arrangement of operations (by parttoolid) :

| partid      | mach | hp | special | load  |
|-------------|------|----|---------|-------|
| 1 00060001  | 1    | 3  | 0       | 0.348 |
| 2 00060002  | 1    | 3  | 0       | 0.069 |
| 3 00060003  | 1    | 1  | 0       | 0.481 |
| 4 00060004  | 1    | 1  | 0       | 0.412 |
| 5 00060010  | 2    | 1  | 0       | 0.417 |
| 6 00060030  | 3    | 3  | 0       | 0.783 |
| 7 00070001  | 1    | 1  | 0       | 0.542 |
| 8 00070002  | 1    | 2  | 0       | 0.748 |
| 9 00070003  | 1    | 3  | 0       | 0.382 |
| 10 00070004 | 1    | 2  | d       | 0.332 |
| 11 00070005 | 1    | 2  | d       | 0.293 |
| 12 00070010 | 2    | 1  | 0       | 0.371 |
| 13 00070011 | 2    | 3  | 0       | 0.259 |
| 14 00070050 | 3    | 3  | 0       | 0.251 |
| 15 00080001 | 1    | 2  | d       | 0.287 |
| 16 00080002 | 1    | 2  | 0       | 0.775 |
| 17 00080003 | 1    | 3  | 0       | 0.475 |
| 18 00080004 | 1    | 1  | 0       | 0.350 |
| 19 00080005 | 1    | 3  | 0       | 0.400 |
| 20 00080006 | 1    | 3  | 0       | 0.325 |
| 21 00080010 | 2    | 3  | 0       | 0.775 |

### current arrangement of operations (by machines) :

|               |                               |        |       |
|---------------|-------------------------------|--------|-------|
| machine 1 1 0 | include parttoolid : 00060003 | load : | 0.481 |
| machine 1 1 0 | include parttoolid : 00060004 | load : | 0.412 |
| machine 1 1 0 | include parttoolid : 00070001 | load : | 0.542 |
| machine 1 1 0 | include parttoolid : 00080004 | load : | 0.350 |
| machine 1 1 0 | total load :                  |        | 1.786 |
|               |                               |        |       |
| machine 1 2 0 | include parttoolid : 00070002 | load : | 0.748 |
| machine 1 2 0 | include parttoolid : 00080002 | load : | 0.775 |
| machine 1 2 0 | total load :                  |        | 1.523 |
|               |                               |        |       |
| machine 1 2 d | include parttoolid : 00070004 | load : | 0.332 |
| machine 1 2 d | include parttoolid : 00070005 | load : | 0.293 |
| machine 1 2 d | include parttoolid : 00080001 | load : | 0.287 |
| machine 1 2 d | total load :                  |        | 0.913 |
|               |                               |        |       |
| machine 1 3 0 | include parttoolid : 00060001 | load : | 0.348 |
| machine 1 3 0 | include parttoolid : 00060002 | load : | 0.069 |
| machine 1 3 0 | include parttoolid : 00070003 | load : | 0.382 |
| machine 1 3 0 | include parttoolid : 00080003 | load : | 0.475 |

|         |   |   |   |                      |          |        |       |
|---------|---|---|---|----------------------|----------|--------|-------|
| machine | 1 | 3 | 0 | include parttoolid : | 00080005 | Load : | 0.400 |
| machine | 1 | 3 | 0 | include parttoolid : | 00080006 | Load : | 0.325 |
| machine | 1 | 3 | 0 | total load :         | 1.999    |        |       |
| machine | 2 | 1 | 0 | include parttoolid : | 00060010 | Load : | 0.417 |
| machine | 2 | 1 | 0 | include parttoolid : | 00070010 | Load : | 0.371 |
| machine | 2 | 1 | 0 | total load :         | 0.788    |        |       |
| machine | 2 | 3 | 0 | include parttoolid : | 00070011 | Load : | 0.259 |
| machine | 2 | 3 | 0 | include parttoolid : | 00080010 | Load : | 0.775 |
| machine | 2 | 3 | 0 | total load :         | 1.034    |        |       |
| machine | 3 | 3 | 0 | include parttoolid : | 00060030 | Load : | 0.783 |
| machine | 3 | 3 | 0 | include parttoolid : | 00070050 | Load : | 0.251 |
| machine | 3 | 3 | 0 | total load :         | 1.035    |        |       |

RUN 3

task number : 13  
rule applied : 35.000

35.000

stage2  
machload(-1,-1,0,1.8,100.0)  
attrload(spoperat,\*, -1,-1,0,1.8,100.0)

HEN

attrtran(spoperat,\*, -1,-1,0,-1,-1,\*,0.5)

urrent existing facts :

1 stage2

urrent arrangement of operations (by parttoolid) :

| partid     | mach | hp | special | load  |
|------------|------|----|---------|-------|
| 1 00010001 | 1    | 2  | d       | 0.364 |
| 2 00010002 | 1    | 3  | c       | 0.556 |
| 3 00010003 | 1    | 3  | o       | 1.750 |
| 4 00010004 | 1    | 2  | o       | 0.519 |
| 5 00010010 | 2    | 3  | m       | 0.294 |
| 6 00010020 | 1    | 3  | o       | 0.103 |
| 7 00020001 | 1    | 3  | o       | 0.719 |
| 8 00020002 | 1    | 2  | d       | 0.244 |
| 9 00020003 | 1    | 2  | t       | 0.381 |
| 0 00020004 | 1    | 2  | b       | 0.483 |
| 1 00020010 | 2    | 3  | m       | 0.364 |
| 2 00020011 | 2    | 1  | f       | 0.400 |
| 3 00020021 | 3    | 2  | o       | 0.656 |
| 4 00020031 | 3    | 3  | o       | 0.453 |
| 5 00020032 | 3    | 2  | o       | 0.659 |
| 6 00030001 | 1    | 3  | o       | 0.625 |
| 7 00030001 | 1    | 1  | o       | 0.604 |
| 8 00030002 | 1    | 3  | c       | 0.438 |
| 9 00030003 | 1    | 2  | t       | 0.415 |
| 0 00030004 | 1    | 2  | b       | 0.463 |
| 1 00030010 | 2    | 1  | f       | 0.229 |
| 2 00030011 | 2    | 3  | m       | 0.104 |
| 3 00040001 | 1    | 3  | o       | 0.229 |
| 4 00040002 | 1    | 3  | c       | 0.094 |
| 5 00040010 | 2    | 1  | f       | 0.151 |
| 6 00050001 | 1    | 3  | o       | 0.669 |
| 7 00050002 | 1    | 2  | o       | 0.463 |
| 8 00050003 | 1    | 2  | t       | 0.151 |
| 9 00050020 | 3    | 3  | o       | 0.204 |
| 0 00060001 | 1    | 3  | o       | 0.348 |
| 1 00060002 | 1    | 3  | o       | 0.069 |
| 2 00060003 | 1    | 1  | o       | 0.481 |
| 3 00060004 | 1    | 1  | o       | 0.412 |
| 4 00060010 | 2    | 1  | f       | 0.208 |
| 5 00060030 | 3    | 3  | o       | 0.783 |
| 6 00070001 | 1    | 1  | o       | 0.542 |
| 7 00070002 | 1    | 2  | o       | 0.748 |
| 8 00070003 | 1    | 3  | o       | 0.382 |
| 9 00070004 | 1    | 2  | d       | 0.332 |
| 0 00070005 | 1    | 2  | d       | 0.293 |
| 1 00070010 | 2    | 1  | f       | 0.186 |
| 2 00070011 | 2    | 3  | m       | 0.129 |

|    |          |   |   |   |       |
|----|----------|---|---|---|-------|
| 43 | 00070050 | 3 | 3 | 0 | 0.251 |
| 44 | 00090001 | 1 | 2 | d | 0.287 |
| 45 | 00080002 | 1 | 2 | t | 0.387 |
| 46 | 00090003 | 1 | 3 | 0 | 0.475 |
| 47 | 00080004 | 1 | 1 | 0 | 0.350 |
| 48 | 00080005 | 1 | 3 | c | 0.200 |
| 49 | 00080006 | 1 | 3 | 0 | 0.325 |
| 50 | 00090010 | 2 | 3 | m | 0.387 |

current arrangement of operations (by machines) :

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 1 | 1 | 0 | include parttoolid : 00030001 | Load : | 0.604 |
| machine | 1 | 1 | 0 | include parttoolid : 00060003 | Load : | 0.481 |
| machine | 1 | 1 | 0 | include parttoolid : 00060004 | Load : | 0.412 |
| machine | 1 | 1 | 0 | include parttoolid : 00070001 | Load : | 0.542 |
| machine | 1 | 1 | 0 | include parttoolid : 00080004 | Load : | 0.350 |
| machine | 1 | 1 | 0 | total load :                  |        | 2.390 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 1 | 2 | 0 | include parttoolid : 00010004 | Load : | 0.519 |
| machine | 1 | 2 | 0 | include parttoolid : 00050002 | Load : | 0.463 |
| machine | 1 | 2 | 0 | include parttoolid : 00070002 | Load : | 0.748 |
| machine | 1 | 2 | 0 | total load :                  |        | 1.729 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 1 | 2 | d | include parttoolid : 00010001 | Load : | 0.364 |
| machine | 1 | 2 | d | include parttoolid : 00020002 | Load : | 0.244 |
| machine | 1 | 2 | d | include parttoolid : 00070004 | Load : | 0.332 |
| machine | 1 | 2 | d | include parttoolid : 00070005 | Load : | 0.293 |
| machine | 1 | 2 | d | include parttoolid : 00080001 | Load : | 0.287 |
| machine | 1 | 2 | d | total load :                  |        | 1.521 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 1 | 2 | t | include parttoolid : 00020003 | Load : | 0.381 |
| machine | 1 | 2 | t | include parttoolid : 00030003 | Load : | 0.415 |
| machine | 1 | 2 | t | include parttoolid : 00050003 | Load : | 0.151 |
| machine | 1 | 2 | t | include parttoolid : 00080002 | Load : | 0.387 |
| machine | 1 | 2 | t | total load :                  |        | 1.334 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 1 | 2 | b | include parttoolid : 00020004 | Load : | 0.483 |
| machine | 1 | 2 | b | include parttoolid : 00030004 | Load : | 0.463 |
| machine | 1 | 2 | b | total load :                  |        | 0.945 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 1 | 3 | 0 | include parttoolid : 00010003 | Load : | 1.750 |
| machine | 1 | 3 | 0 | include parttoolid : 00010020 | Load : | 0.103 |
| machine | 1 | 3 | 0 | include parttoolid : 00020001 | Load : | 0.719 |
| machine | 1 | 3 | 0 | include parttoolid : 00030001 | Load : | 0.625 |
| machine | 1 | 3 | 0 | include parttoolid : 00040001 | Load : | 0.229 |
| machine | 1 | 3 | 0 | include parttoolid : 00050001 | Load : | 0.669 |
| machine | 1 | 3 | 0 | include parttoolid : 00060001 | Load : | 0.348 |
| machine | 1 | 3 | 0 | include parttoolid : 00060002 | Load : | 0.069 |
| machine | 1 | 3 | 0 | include parttoolid : 00070003 | Load : | 0.382 |
| machine | 1 | 3 | 0 | include parttoolid : 00080003 | Load : | 0.475 |
| machine | 1 | 3 | 0 | include parttoolid : 00080006 | Load : | 0.325 |
| machine | 1 | 3 | 0 | total load :                  |        | 5.694 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 1 | 3 | c | include parttoolid : 00010002 | Load : | 0.556 |
| machine | 1 | 3 | c | include parttoolid : 00030002 | Load : | 0.438 |
| machine | 1 | 3 | c | include parttoolid : 00040002 | Load : | 0.094 |
| machine | 1 | 3 | c | include parttoolid : 00080005 | Load : | 0.200 |
| machine | 1 | 3 | c | total load :                  |        | 1.288 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 2 | 1 | f | include parttoolid : 00020011 | Load : | 0.400 |
| machine | 2 | 1 | f | include parttoolid : 00030010 | Load : | 0.229 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 2 | 1 | f | include parttoolid : 00040010 | load : | 0.151 |
| machine | 2 | 1 | f | include parttoolid : 00060010 | load : | 0.208 |
| machine | 2 | 1 | f | include parttoolid : 00070010 | load : | 0.186 |
| machine | 2 | 1 | f | total load :                  |        | 1.174 |
| machine | 2 | 3 | m | include parttoolid : 00010010 | load : | 0.294 |
| machine | 2 | 3 | m | include parttoolid : 00020010 | load : | 0.364 |
| machine | 2 | 3 | m | include parttoolid : 00030011 | load : | 0.104 |
| machine | 2 | 3 | m | include parttoolid : 00070011 | load : | 0.129 |
| machine | 2 | 3 | m | include parttoolid : 00080010 | load : | 0.387 |
| machine | 2 | 3 | m | total load :                  |        | 1.279 |
| machine | 3 | 2 | 0 | include parttoolid : 00020021 | load : | 0.656 |
| machine | 3 | 2 | 0 | include parttoolid : 00020032 | load : | 0.659 |
| machine | 3 | 2 | 0 | total load :                  |        | 1.316 |
| machine | 3 | 3 | 0 | include parttoolid : 00020031 | load : | 0.453 |
| machine | 3 | 3 | 0 | include parttoolid : 00050020 | load : | 0.204 |
| machine | 3 | 3 | 0 | include parttoolid : 00060030 | load : | 0.783 |
| machine | 3 | 3 | 0 | include parttoolid : 00070050 | load : | 0.251 |
| machine | 3 | 3 | 0 | total load :                  |        | 1.692 |

# RUN 4

- 260 -

stack number : 23  
rule applied : 35.000

35.000

1

stage2  
machload(-1,-1,0,1.8,100.0)  
attrload(spooperat,\*,-1,-1,0,1.8,100.0)

THEN

attrtran(spooperat,\*,-1,-1,0,-1,-1,\*,0.5)

current existing facts :

1 stage2

current arrangement of operations (by parttoolid) :

| partid      | mach | hp | special | load  |
|-------------|------|----|---------|-------|
| 1 00010001  | 1    | 2  | d       | 0.364 |
| 2 00010002  | 1    | 3  | c       | 0.556 |
| 3 00010003  | 1    | 3  | 0       | 1.750 |
| 4 00010004  | 1    | 2  | 0       | 0.519 |
| 5 00010010  | 2    | 3  | m       | 0.294 |
| 6 00010020  | 1    | 3  | 0       | 0.103 |
| 7 00020001  | 1    | 3  | 0       | 0.719 |
| 8 00020002  | 1    | 2  | d       | 0.244 |
| 9 00020003  | 1    | 2  | t       | 0.381 |
| 10 00020004 | 1    | 2  | b       | 0.483 |
| 11 00020010 | 2    | 3  | m       | 0.364 |
| 12 00020011 | 2    | 1  | f       | 0.400 |
| 13 00020021 | 3    | 2  | 0       | 0.656 |
| 14 00020031 | 3    | 3  | 0       | 0.453 |
| 15 00020032 | 3    | 2  | 0       | 0.659 |
| 16 00030001 | 1    | 3  | 0       | 0.625 |
| 17 00030001 | 1    | 3  | 0       | 0.604 |
| 18 00030002 | 1    | 3  | c       | 0.438 |
| 19 00030003 | 1    | 2  | t       | 0.415 |
| 20 00030004 | 1    | 2  | b       | 0.463 |
| 21 00030010 | 2    | 1  | f       | 0.229 |
| 22 00030011 | 2    | 3  | m       | 0.104 |
| 23 00040001 | 1    | 3  | 0       | 0.229 |
| 24 00040002 | 1    | 3  | c       | 0.094 |
| 25 00040010 | 2    | 1  | f       | 0.151 |
| 26 00050001 | 1    | 3  | 0       | 0.669 |
| 27 00050002 | 1    | 2  | 0       | 0.463 |
| 28 00050003 | 1    | 2  | t       | 0.151 |
| 29 00050020 | 3    | 3  | 0       | 0.204 |
| 30 00060001 | 1    | 3  | 0       | 0.348 |
| 31 00060002 | 1    | 3  | 0       | 0.069 |
| 32 00060003 | 1    | 1  | 0       | 0.481 |
| 33 00060004 | 1    | 1  | 0       | 0.412 |
| 34 00060010 | 2    | 1  | f       | 0.208 |
| 35 00060030 | 3    | 3  | 0       | 0.783 |
| 36 00070001 | 1    | 1  | 0       | 0.542 |
| 37 00070002 | 1    | 2  | 0       | 0.748 |
| 38 00070003 | 1    | 3  | 0       | 0.382 |
| 39 00070004 | 1    | 2  | d       | 0.332 |
| 40 00070005 | 1    | 2  | d       | 0.293 |
| 41 00070010 | 2    | 1  | f       | 0.186 |
| 42 00070011 | 2    | 3  | m       | 0.129 |



|    |          |   |   |   |       |
|----|----------|---|---|---|-------|
| 3  | 00070050 | 3 | 3 | 0 | 0.251 |
| 4  | 00080001 | 1 | 2 | d | 0.287 |
| 5  | 00080002 | 1 | 2 | 0 | 0.775 |
| 6  | 00080003 | 1 | 3 | 0 | 0.475 |
| 7  | 00080004 | 1 | 1 | 0 | 0.350 |
| 8  | 00080005 | 1 | 3 | 0 | 0.400 |
| 9  | 00080006 | 1 | 3 | 0 | 0.325 |
| 10 | 00080010 | 2 | 3 | m | 0.387 |

Current arrangement of operations (by machines) :

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| Machine | 1 | 1 | 0 | include parttoolid : 00060003 | Load : | 0.481 |
| Machine | 1 | 1 | 0 | include parttoolid : 00060004 | Load : | 0.412 |
| Machine | 1 | 1 | 0 | include parttoolid : 00070001 | Load : | 0.542 |
| Machine | 1 | 1 | 0 | include parttoolid : 00080004 | Load : | 0.350 |
| Machine | 1 | 1 | 0 | total load :                  |        | 1.786 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| Machine | 1 | 2 | 0 | include parttoolid : 00010004 | Load : | 0.519 |
| Machine | 1 | 2 | 0 | include parttoolid : 00050002 | Load : | 0.463 |
| Machine | 1 | 2 | 0 | include parttoolid : 00070002 | Load : | 0.748 |
| Machine | 1 | 2 | 0 | include parttoolid : 00080002 | Load : | 0.775 |
| Machine | 1 | 2 | 0 | total load :                  |        | 2.504 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| Machine | 1 | 2 | d | include parttoolid : 00010001 | Load : | 0.364 |
| Machine | 1 | 2 | d | include parttoolid : 00020002 | Load : | 0.244 |
| Machine | 1 | 2 | d | include parttoolid : 00070004 | Load : | 0.332 |
| Machine | 1 | 2 | d | include parttoolid : 00070005 | Load : | 0.293 |
| Machine | 1 | 2 | d | include parttoolid : 00080001 | Load : | 0.287 |
| Machine | 1 | 2 | d | total load :                  |        | 1.521 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| Machine | 1 | 2 | t | include parttoolid : 00020003 | Load : | 0.381 |
| Machine | 1 | 2 | t | include parttoolid : 00030003 | Load : | 0.415 |
| Machine | 1 | 2 | t | include parttoolid : 00050003 | Load : | 0.151 |
| Machine | 1 | 2 | t | total load :                  |        | 0.947 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| Machine | 1 | 2 | b | include parttoolid : 00020004 | Load : | 0.483 |
| Machine | 1 | 2 | b | include parttoolid : 00030004 | Load : | 0.463 |
| Machine | 1 | 2 | b | total load :                  |        | 0.945 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| Machine | 1 | 3 | 0 | include parttoolid : 00010003 | Load : | 1.750 |
| Machine | 1 | 3 | 0 | include parttoolid : 00010020 | Load : | 0.103 |
| Machine | 1 | 3 | 0 | include parttoolid : 00020001 | Load : | 0.719 |
| Machine | 1 | 3 | 0 | include parttoolid : 00030001 | Load : | 0.625 |
| Machine | 1 | 3 | 0 | include parttoolid : 00030001 | Load : | 0.604 |
| Machine | 1 | 3 | 0 | include parttoolid : 00040001 | Load : | 0.229 |
| Machine | 1 | 3 | 0 | include parttoolid : 00050001 | Load : | 0.669 |
| Machine | 1 | 3 | 0 | include parttoolid : 00060001 | Load : | 0.348 |
| Machine | 1 | 3 | 0 | include parttoolid : 00060002 | Load : | 0.069 |
| Machine | 1 | 3 | 0 | include parttoolid : 00070003 | Load : | 0.382 |
| Machine | 1 | 3 | 0 | include parttoolid : 00080003 | Load : | 0.475 |
| Machine | 1 | 3 | 0 | include parttoolid : 00080005 | Load : | 0.400 |
| Machine | 1 | 3 | 0 | include parttoolid : 00080006 | Load : | 0.325 |
| Machine | 1 | 3 | 0 | total load :                  |        | 6.698 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| Machine | 1 | 3 | c | include parttoolid : 00010002 | Load : | 0.556 |
| Machine | 1 | 3 | c | include parttoolid : 00030002 | Load : | 0.438 |
| Machine | 1 | 3 | c | include parttoolid : 00040002 | Load : | 0.094 |
| Machine | 1 | 3 | c | total load :                  |        | 1.087 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| Machine | 2 | 1 | f | include parttoolid : 00020011 | Load : | 0.400 |
| Machine | 2 | 1 | f | include parttoolid : 00030010 | Load : | 0.229 |

26

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 2 | 1 | f | include parttoolid : 00040010 | load : | 0.151 |
| machine | 2 | 1 | f | include parttoolid : 00060010 | load : | 0.208 |
| machine | 2 | 1 | f | include parttoolid : 00070010 | load : | 0.186 |
| machine | 2 | 1 | f | total load :                  |        | 1.174 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 2 | 3 | m | include parttoolid : 00010010 | load : | 0.294 |
| machine | 2 | 3 | m | include parttoolid : 00020010 | load : | 0.364 |
| machine | 2 | 3 | m | include parttoolid : 00030011 | load : | 0.104 |
| machine | 2 | 3 | m | include parttoolid : 00070011 | load : | 0.129 |
| machine | 2 | 3 | m | include parttoolid : 00080010 | load : | 0.387 |
| machine | 2 | 3 | m | total load :                  |        | 1.279 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 3 | 2 | 0 | include parttoolid : 00020021 | load : | 0.656 |
| machine | 3 | 2 | 0 | include parttoolid : 00020032 | load : | 0.659 |
| machine | 3 | 2 | 0 | total load :                  |        | 1.316 |

|         |   |   |   |                               |        |       |
|---------|---|---|---|-------------------------------|--------|-------|
| machine | 3 | 3 | 0 | include parttoolid : 00020031 | load : | 0.453 |
| machine | 3 | 3 | 0 | include parttoolid : 00050020 | load : | 0.204 |
| machine | 3 | 3 | 0 | include parttoolid : 00060030 | load : | 0.783 |
| machine | 3 | 3 | 0 | include parttoolid : 00070050 | load : | 0.251 |
| machine | 3 | 3 | 0 | total load :                  |        | 1.692 |

# RULES

- 263 -

```
0.9
IF
stage0
nofactex(define)
ENDIF
THEN
defoptyp(3)
deihptyp(3)
define
ENDTHEN
1
IF
attrexis(opertype,1,0,0,0)
ENDIF
THEN
attrtran(opertype,1,0,0,0,1,0,0,1.0)
ENDTHEN
2
IF
attrexis(opertype,2,0,0,0)
ENDIF
THEN
attrtran(opertype,2,0,0,0,2,0,0,1.0)
ENDTHEN
3
IF
attrexis(opertype,3,0,0,0)
ENDIF
THEN
attrtran(opertype,3,0,0,0,3,0,0,1.0)
ENDTHEN
4
IF
attrexis(opertype,4,0,0,0)
ENDIF
THEN
attrtran(opertype,4,0,0,0,3,0,0,1.0)
ENDTHEN
5
IF
attrexis(opertype,5,0,0,0)
ENDIF
THEN
attrtran(opertype,5,0,0,0,3,0,0,1.0)
ENDTHEN
6
forallopty
IF
nofactex(stage1)
machexis(-1,0,0)
ENDIF
THEN
delefact(stage0)
delefact(define)
stage1
ENDTHEN
7
forallopty
IF
stage1
```

```
machaxis(-1,0,0)
attrexaxis(houpower,0.0,10.0,-1,0,0)
ENDIF
THEN
attrtran(houpower,0.0,10.0,-1,0,0,-1,1,0,1.0)
ENDTHEN
8
foralllopty
IF
stage1
attrexaxis(houpower,10.0,20.0,-1,0,0)
ENDIF
THEN
attrtran(houpower,10.0,20.0,-1,0,0,-1,2,0,1.0)
ENDTHEN
9
foralllopty
IF
stage1
attrexaxis(houpower,20.0,1000.0,-1,0,0)
ENDIF
THEN
attrtran(houpower,20.0,1000.0,-1,0,0,-1,3,0,1.0)
ENDTHEN
10
IF
stage1
ENDIF
THEN
delefact(stage1)
stage2
ENDTHEN
11
forallhpty
IF
stage2
machaxis(1,-1,0)
machload(1,-1,0,0.0,0.7)
machload(2,-1,0,0.3,100.0)
ENDIF
THEN
transfer(1,-1,0,2,-1,0,1.0)
ENDTHEN
12
forallhpty
IF
stage2
machaxis(2,-1,0)
machload(2,-1,0,0.0,0.7)
machload(3,-1,0,0.3,100.0)
ENDIF
THEN
transfer(2,-1,0,3,-1,0,1.0)
ENDTHEN
13
forallhpty
IF
stage2
machaxis(3,-1,0)
machload(3,-1,0,0.0,0.7)
```

```
machload(4,-1,0,0.3,100.0)
ENDIF
THEN
transfer(3,-1,0,4,-1,0,1.0)
ENDTHEN
14
forallhpty
IF
stage2
machaxis(4,-1,0)
machload(4,-1,0,0.0,0.7)
machload(5,-1,0,0.3,100.0)
ENDIF
THEN
transfer(4,-1,0,5,-1,0,1.0)
ENDTHEN
16
IF
stage2
machaxis(1,1,0)
machload(1,1,0,0.0,0.7)
machload(2,1,0,0.0,0.3)
machload(1,2,0,0.3,100.0)
ENDIF
THEN
transfer(1,1,0,1,2,0,1.0)
ENDTHEN
17
IF
stage2
machaxis(1,2,0)
machload(1,2,0,0.0,0.7)
machload(2,2,0,0.0,0.3)
machload(1,3,0,0.3,100.0)
ENDIF
THEN
transfer(1,2,0,1,3,0,1.0)
ENDTHEN
18
IF
stage2
machaxis(1,3,0)
machload(1,3,0,0.0,0.7)
machload(2,3,0,0.0,0.3)
machload(1,4,0,0.3,100.0)
ENDIF
THEN
transfer(1,3,0,1,4,0,1.0)
ENDTHEN
19
IF
stage2
machaxis(1,4,0)
machload(1,4,0,0.0,0.7)
machload(2,4,0,0.0,0.3)
machload(1,5,0,0.3,100.0)
ENDIF
THEN
transfer(1,4,0,1,5,0,1.0)
ENDTHEN
```

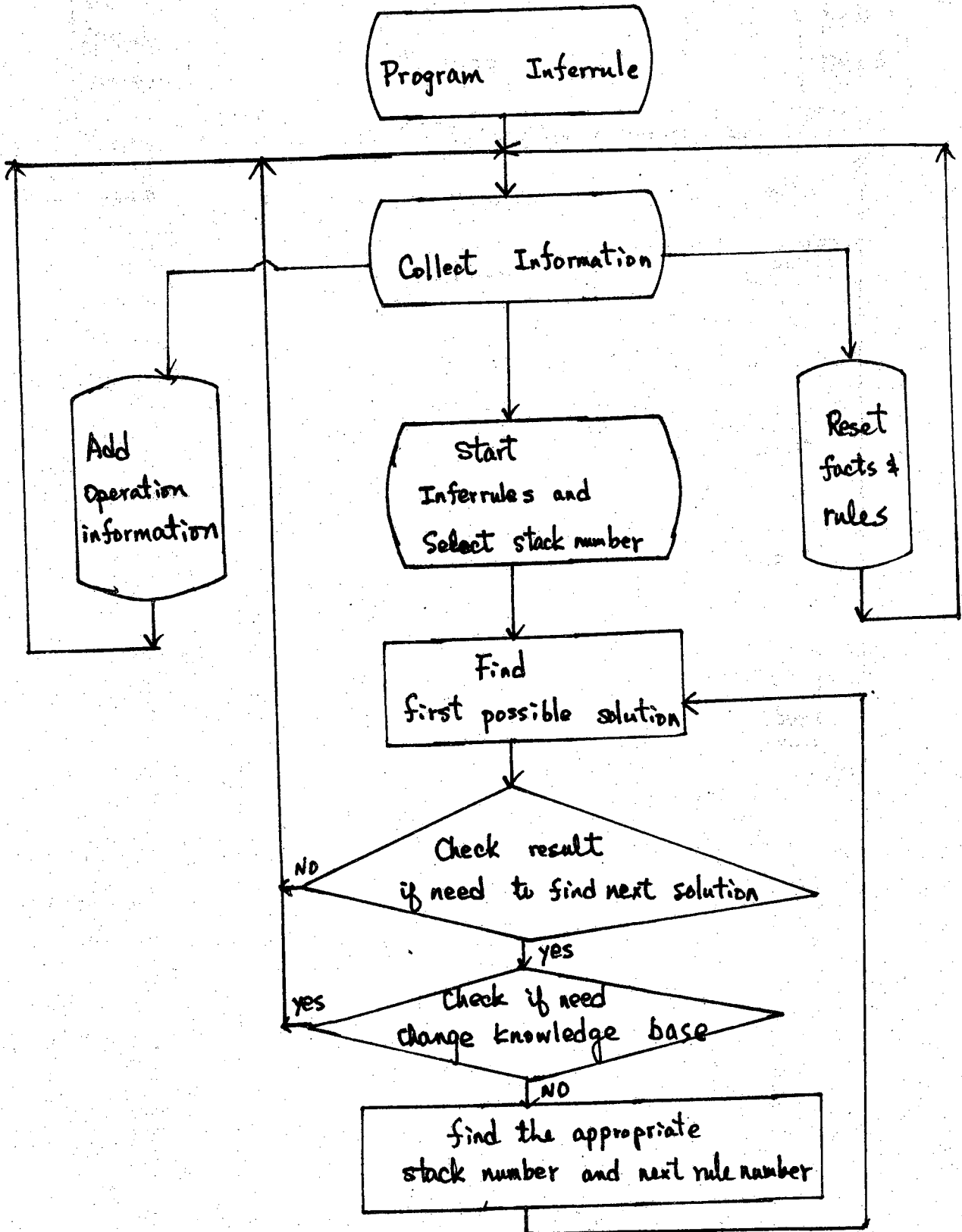
```
21
IF
stage2
machaxis(2,1,0)
machload(2,1,0,0.0,0.7)
machload(3,1,0,0.0,0.3)
machload(2,2,0,0.3,100.0)
ENDIF
THEN
transfer(2,1,0,2,2,0,1.0)
ENDTHEN
22
IF
stage2
machaxis(2,2,0)
machload(2,2,0,0.0,0.7)
machload(3,2,0,0.0,0.3)
machload(2,3,0,0.3,100.0)
ENDIF
THEN
transfer(2,2,0,2,3,0,1.0)
ENDTHEN
23
IF
stage2
machaxis(2,3,0)
machload(2,3,0,0.0,0.7)
machload(3,3,0,0.0,0.3)
machload(2,4,0,0.3,100.0)
ENDIF
THEN
transfer(2,3,0,2,4,0,1.0)
ENDTHEN
24
IF
stage2
machaxis(2,4,0)
machload(2,4,0,0.0,0.7)
machload(3,4,0,0.0,0.3)
machload(2,5,0,0.3,100.0)
ENDIF
THEN
transfer(2,4,0,2,5,0,1.0)
ENDTHEN
25
IF
stage2
machaxis(3,1,0)
machload(3,1,0,0.0,0.7)
machload(4,1,0,0.0,0.3)
machload(3,2,0,0.3,100.0)
ENDIF
THEN
transfer(3,1,0,3,2,0,1.0)
ENDTHEN
26
IF
stage2
machaxis(3,2,0)
machload(3,2,0,0.0,0.7)
```

```
machload(4,2,0,0.0,0.3)
machload(3,3,0,0.3,100.0)
ENDIF
THEN
transfer(3,2,0,3,3,0,1.0)
ENDTHEN
27
IF
stage2
machexis(3,3,0)
machload(3,3,0,0.0,0.7)
machload(4,3,0,0.0,0.3)
machload(3,4,0,0.3,100.0)
ENDIF
THEN
transfer(3,3,0,3,4,0,1.0)
ENDTHEN
28
IF
stage2
machexis(3,4,0)
machload(3,4,0,0.0,0.7)
machload(4,4,0,0.0,0.3)
machload(3,5,0,0.3,100.0)
ENDIF
THEN
transfer(3,4,0,3,5,0,1.0)
ENDTHEN
30
IF
stage2
machexis(4,1,0)
machload(4,1,0,0.0,0.7)
machload(5,1,0,0.0,0.3)
machload(4,2,0,0.3,100.0)
ENDIF
THEN
transfer(4,1,0,4,2,0,1.0)
ENDTHEN
31
IF
stage2
machexis(4,2,0)
machload(4,2,0,0.0,0.7)
machload(5,2,0,0.0,0.3)
machload(4,3,0,0.3,100.0)
ENDIF
THEN
transfer(4,2,0,4,3,0,1.0)
ENDTHEN
32
IF
stage2
machexis(4,3,0)
machload(4,3,0,0.0,0.7)
machload(5,3,0,0.0,0.3)
machload(4,4,0,0.3,100.0)
ENDIF
THEN
transfer(4,3,0,4,4,0,1.0)
```

```
ENDTHEN
33
IF
stage2
machaxis(4,4,0)
machload(4,4,0,0.0,0.7)
machload(5,4,0,0.0,0.3)
machload(4,5,0,0.3,100.0)
ENDIF
THEN
transfer(4,4,0,4,5,0,1.0)
ENDTHEN
35
foralloty
forallhpty
forallspop
IF
stage2
machload(-1,-1,0,1.8,100.0)
attrload(soperat,*,-1,-1,0,1.8,100.0)
ENDIF
THEN
attrtran(soperat,*,-1,-1,0,-1,-1,* ,0.5)
ENDTHEN
```



# Program Flowchart



---

**Chapter 6**

**PROLOG EXPERT: A Simple PROLOG based Expert System Framework for  
Synthesis, the BAGGER Problem as An Example**

*T. Sarjakoski*

PROLOG EXPERT: A simple PROLOG based Expert System framework for synthesis, the BAGGER problem as an example

Tapani Sarjakoski

## 1. INTRODUCTION

The design of this system is motivated by my research "EXPERT SYSTEM FOR ANALYZING PHOTOGRAMMETRIC MEASUREMENT DATA": I noticed that there is a great resemblance between my research and the BAGGER. The BAGGER suited well as an example problem to develop a simple EXPERT SYSTEM framework.

As stated above, my research deals with analyzing photogrammetric measurement data. On the other hand, the BAGGER is an example of synthesis. How can there be anything in common? There is, because the task of analyzing photogrammetric measurement data involves also synthesis: The task is very much computation oriented and it is thus critical that there is a plan of the sequence in which the computations will be carried out. On the other hand, the BAGGER also includes some features of an analysis system: it has to decide, for example, if a bag is already full.

In the following I have not expanded the BAGGER example as such but I have designed a simple PROLOG based framework for synthesis. It is not limited for BAGGER example and could be used for example to implement my own problem.

## 2. SYSTEM ORGANIZATION

### 2.1. Remarks of PROLOG and its usage

The PROLOG is as such already a fairly powerful tool for Logic Programming: Facts and rules can be entered in a straightforward manner because the PROLOG interpreted is actually a realization of a system utilizing Clausal Form of Logic.

The first implementation of the BAGGER is a good example of using PROLOG directly for building an Expert System. The subsequent phases of the BAGGER can be realized as a simple PROLOG clause:

```
run_bagger :-  
    check_order_phase,  
    bag_large_items_phase,  
    bag_medium_items_phase,  
    bag_small_items_phase.
```

The context limiting is now realized in a very simple way: by writing a procedure for each phase. Because PROLOG interpreter satisfies the goals from left to right it actually assures a fixed order of execution.

The complete listing of the first implementation of the BAGGER is in Appendix BAGGER1. I consider that it shows very clearly that an Expert System can be written in PROLOG very easily by using a conventional procedural approach. However, there is missing one feature which is usually considered to be an essential part of an Expert System: an Explanation Facility.

In the following is described another PROLOG approach which is called PROLOG EXPERT. It is a kind of framework which is also supported by a simple explanation facility, "explain".

## 2.2. Knowledge representation in PROLOG EXPERT

### 2.2.1. Facts

The facts are represented by using PROLOG clauses, for example:

```
unbagged(bread).  
container(bread,plastic_bag).  
size(granola,large).  
frozen(bread,no).  
frozen(ice_cream,yes).
```

All the facts are considered to be located on the BLACK BOARD of the PROLOG EXPERT".

The total set of facts for the BAGGER example are listed in the Appendix BAGGER FACTS.

### 2.2.2. Rules

There are two kinds of rules which can be used in PROLOG EXPERT: main rules and auxiliary rules.

Main rules are written as PROLOG clauses and have the syntax

```
rule(<rulename>) :- <conditions>,then,<actions>.
```

for example

```
rule(b1) :-
```

```
step(check_order),
unbagged(potato_chips),
soft_drink(X),
not unbagged(X),
then,
add_unbagged(pepsi).
```

The inference machine has access to this rules

Auxiliary rules are also written as PROLOG clauses. As a matter of fact they can be any PROLOG clauses. The auxiliary rules are accessed by the main rules, either as conditions or as actions, for example in the rule(b1) we have used the auxiliary rule

```
add_unbagged(X) :-
write_on_black_board(unbagged(X)),
printstring(" < "),
print(X),
printstring(" added ..."),nl.
```

Note: The print-commands are included only for offering some progress information for demonstrating the run of the PROLOG EXPERT in solving the BAGGER example.

If some auxiliary rules are used as conditions they can be interpreted as "derived facts".

There are two important procedures which must be used in the action parts of the main rules (or in the auxiliary rules if these are then used in the action parts):

```
write_on_black_board(<fact>)
and
remove_from(<fact>).
```

They allow the user to add facts to or remove facts from the BLACK BOARD.

### 2.2.3. Explanations

For each main rule the user has to enter also an explanation. The syntax for it is as follows:

```
explanation(<rulename>,X) :-
X = [
["if",
" <explanation for condition 1>",
.
.
" <explanation for condition n>"],
["then",
" <explanation for action 1>".
```

"<explanation for action m>"]].

The explanation is used for clarifying the meaning of the rule in English. It can be displayed with the Explanator Facility.

All the main rules, auxiliary rules and explanations are listed in Appendix BAGGER RULES.

### 2.3. Inference Machine

The Inference Machine of the PROLOG EXPERT would be really simple if no history would be collected of the application of the rules:

```
prolog_expert :-
    repeat
    apply_rule(Rule_name),
    step(stop).

apply_rule(Rule_name) :-
    rule(Rule_name), !.
```

What actually would happen when using this Inference Machine, is that goals

```
rule(Rule_name) and
step(stop)
```

would be tried to be satisfied repeatedly until the goal

```
step(stop)
```

is satisfied. This means that the user can stop the Inference Machine by adding the fact

```
step(stop).
```

to the BLACK BOARD.

Because of the way PROLOG interpreted searches through the data base, "prolog\_expert" will always apply Rule Ordering for Conflict Resolution. It must be noticed, however, that Specificity Ordering and Size Ordering can actually be implemented by using Rule Ordering - simply by ordering the rules properly in the data base.

The actual implementation (listed in Appendix PROLOG EXPERT) is only slightly more complex because it generates a list of the application of rules: Following facts are saved each time a rule fires:

- rule name
- list of facts removed from the BLACK BOARD
- list of facts added to the BLACK BOARD

The history list is stored as a fact on the BLACK BOARD. Additionally, two more facts, "additions" and "removals", are used to pass the information of actions on BLACK BOARD from procedures "write\_on\_black\_board" and "remove\_from" to the procedure "up\_date\_history\_list".

The Inference Machine does not control the application of the auxiliary rules, nor does it collect any history of their application.

#### 2.4. Explanation facility

Once the PROLOG EXPERT has been run, the history of the application of the rules can be investigated by using the explanation facility "explain". It allows the user to scan the history list forwards and backwards. For each application of a rule it is possible to display the actions on the BLACK BOARD as well as the explanation and the rule itself.

The explanation facility is listed in Appendix PROLOG EXPERT/EXPLAIN. Additionally, its functions will become very obvious by looking at the example run at Appendix EXAMPLE RUN.

### 3. EDITING RULES AND FACTS

The rules and facts are stored in normal text files. They can be manipulated by conventional text editors. No special system is designed for entering rules and facts. I feel that it is quite an adequate solution. Especially, because the rules and facts can be split into several files according to the context. The string-search capability of the editors is very useful. The user has to take care of the proper ordering of the rules.

### 4. RUNNING THE SYSTEM

For running the system it is necessary to start the PROLOG interpreter and load the PROLOG EXPERT as well as the case-dependent facts and rules into the PROLOG data base.

For the BAGGER example following parts must be loaded:

- general routines, like "append", "member", "print-string"
- "prolog\_expert" and "prolog\_expert/explain"
- the facts and rules of the problem

-the initial facts which also have to be on the BLACK BOARD (the ones of BAGGER are listed in Appendix INITIAL FACTS)

The possible syntax errors of the rules and facts will be discovered by the PROLOG interpreter.

The problem can be solved by keying in the goal "prolog\_expert". The output of the program is listed in Appendix EXAMPLE RUN.

Once the program has been finished, the final state on the BLACK BOARD can be investigated by using for example PROLOG commands:

listing(initiated\_bag).

listing(number\_of\_bags).

listing(unbagged).

At the moment, no more sophisticated tools have been built for that purpose.

The explanation facility can be started by keying in the goal "explain". The progress of the explain facility is very much self-explanatory which can be noticed from Appendix EXAMPLE RUN.

## 5. DISCUSSION

### 5.1. PROLOG as a basis for building an EXPERT SYSTEM

As pointed out already in the introduction, PROLOG is a very suitable tool for building an problem solver with an essence of an Expert System. This is demonstrated by BAGGER1. In a straightforward approach no explanation facility is offered.

As the implementation of PROLOG EXPERT shows, it is relatively easy to design a simple framework for rule-based Expert System with a simple explanation facility. However, the framework obviously poses some restrictions. For example, the contribution of the "auxiliary rules" on the solution of the problem does not show up in the history. It is possible to replace the "auxiliary rules" by writing corresponding "main rules" which add the derived facts - or rules - to the BLACK BOARD. But then we lose something of the expressive power of PROLOG: It is very natural to express the derived facts in PROLOG as Horn clauses, having the conclusion on the left hand side and the conditions on the right hand side.



### 5.2. Compromise between Expert System and procedural approaches

It seems beneficial to implement some portions of an Expert System by using conventional procedural approach. This holds especially for the low-level operations of an Expert System, for example in the BAGGER it was very natural to write the procedure (or auxiliary rule) "start\_fresh\_bag", even if it would be possible to implement it by using main rules. Analogically, it is very natural to write PROLOG clause for "full bag".

These low level rules (derived facts, rules and actions) can be called "elementary conditions" and "elementary actions". They would, on a way, define a case-dependent language, which is reasonably static for the problem. It also has to be well defined, for example the BLACK BOARD actions must be well known. Also, all the people being involved with the case have to know and understand the elementary operations.

### 5.3. More about the Explanation Facility

For clarifying the explanations, I have used the technique of writing separate explanations for each rule. I consider it useful, it allows the usage of liberal, comment-like explanations. Of course, there is a danger that the explanations do not correspond to the actual rules. Lack of explaining the application of the auxiliary rules is a drawback in the current Explanation Facility.

### 5.4. Interaction

The current BAGGER runs as a pure batch process because all the facts are considered to be given at the moment of starting the BAGGER. It would be possible to add rules for asking for more facts. In such a situation a more versatile Explanation Facility would be useful for answering to questions like WHY?.

### 5.5. Looping for ever

In the current realization of PROLOG BAGGER there is a possibility that the program will run in a loop for ever, due to improper set of rules for solving the problem. It would be relatively simple to expand PROLOG EXPERT for covering that situation: If no rule fires or the firing rule has not caused any changes on the BLACK BOARD, the program is in an infinite loop and must be interrupted. Automatic entering of the Explanation Facility and the user possibility to modify the facts would be a nice feature for debugging purposes.

APPENDIX A example run of the first version  
for solving the BAGGER problem

Script started on Sat Jan 5 18:55:03 1985

? prolog

CProlog version 1.3

| ?- [start].

editor consulted 576 bytes 0.116667 sec.

append2 consulted 120 bytes 0.0666671 sec.

member1 consulted 148 bytes 0.0500008 sec.

start consulted 844 bytes 0.316667 sec.

yes

| ?- [baggl].

baggl consulted 7600 bytes 1.6 sec.

yes

| ?- run\_bagger.

Bagger starts checking the order

< pepsi > added into the list of unbagged items

Bagger starts bagging large items

< pepsi > is placed in bag number < 1 >

< granola > is placed in bag number < 1 >

< granola > is placed in bag number < 1 >

Bagger starts bagging medium items

< ice\_cream > is put in an insulated freezer bag

< bread > is placed in bag number < 2 >

< ice\_cream > is placed in bag number < 2 >

< potato\_chips > is placed in bag number < 2 >

Bagger starts bagging small items

< glop > is placed in bag number < 2 >

yes

| ?- cat(baggl).

unbagged(bread).

unbagged(glop).

unbagged(granola).

unbagged(granola).

unbagged(ice\_cream).

unbagged(potato\_chips).

container(bread,plastic\_bag).

container(glop,jar).

container(granola,cardboard\_box).

container(ice\_cream,cardbord\_carton).

container(pepsi,bottle).

container(potato\_chips,plastic\_bag).

```
size(bread,medium).
size(glop,small).
size(granola,large).
size(ice_cream,medium).
size(pepsi,large).
size(potato_chips,medium).
```

```
frozen(bread,no).
frozen(glop,no).
frozen(granola,no).
frozen(ice_cream,yes).
frozen(pepsi,no).
frozen(potato_chips,no).
```

```
soft_drink(pepsi).
```

```
run_bagger :-
    check_order_phase,
    bag_large_items_phase,
    bag_medium_items_phase,
    bag_small_items_phase,!.

```

```
check_order_phase :-
    nl,
    printstring("Bagger starts checking the order"),nl,
    nl,
    check_order..

```

```
check_order :-
    unbagged(potato_chips),
    soft_drink(X),
    not unbagged(X),
    add_unbagged(pepsi),
    check_order.

```

/\* rule B1 \*/

```
check_order :- !.

```

/\* rule B2 \*/

```
add_unbagged(X) :-
    asserta(unbagged(X)),
    printstring(" < "),
    print(X),
    printstring(" > added into the list of unbagged items")

```

```
check_selection :- !. /* this is a dummy rule for assuring that */
/* 'check_selection' does not fail */

```

```
bag_large_items_phase :-
    nl,
    printstring("Bagger starts bagging large items"),rl,
    nl,
    bag_large_items, !.

```

```
bag_medium_items_phase :-
    nl,
    printstring("Bagger starts bagging medium items"),nl,
    nl,
    bag_medium_items,
    !.

```

```
bag_small_items_phase :-

```

```

nl,
printstring("Bagger starts bagging small items"),rl,
nl,
bag_small_items,
!.

```

```

bag_large_items :-                                     /* rule B3 */
  unbagged(X),
  size(X,large),
  container(X,bottle),
  items_in_bag(Bag_number,large,Number),
  Number < 6,
  put_item_in_bag(X,Bag_number),
  bag_large_items.

```

```

bag_large_items :-                                     /* rule B4 */
  unbagged(X),
  size(X,large),
  items_in_bag(Bag_number,large,Number),
  Number < 6,
  put_item_in_bag(X,Bag_number),
  bag_large_items.

```

```

bag_large_items :-                                     /* rule B5 */
  unbagged(X),
  size(X,large),
  start_fresh_bag,
  bag_large_items.

```

```

bag_large_items :- !.                                 /* rule B6 */

```

```

bag_medium_items :-                                   /* rule B7 */
  unbagged(X),
  size(X,medium),
  empty_bag_or_bag_with_medium_items(Bag_number),
  not_full_bag(Bag_number),
  frozen(X,yes),
  not_insulated(X),
  put_in_insulated_freezer_bag(X),
  bag_medium_items.

```

```

bag_medium_items :-                                   /* rule B8 */
  unbagged(X),
  size(X,medium),
  empty_bag_or_bag_with_medium_items(Bag_number),
  not_full_bag(Bag_number),
  put_item_in_bag(X,Bag_number),
  bag_medium_items.

```

```

bag_medium_items :-                                   /* rule B9 */
  unbagged(X),
  size(X,medium),
  start_fresh_bag,
  bag_medium_items.

```

```

bag_medium_items :- !.                                 /* rule B10 */

```

```

bag_small_items :-                                    /* rule B11 */
  unbagged(X),
  size(X,small),

```

```

not_full_bag(Bag_number),
not_contains(Bag_number,bottle),
put_item_in_bag(X,Bag_number),
bag_small_items.

```

```

bag_small_items :-                               /* rule B12 */
    unbagged(X),
    size(X,small),
    not_full_bag(Bag_number),
    put_item_in_bag(X,Bag_number),
    bag_small_items.

```

```

bag_small_items :-                               /* rule B13 */
    unbagged(X),
    size(X,small),
    start_fresh_bag,
    bag_small_items.

```

```

bag_small_items :- !.                            /* rule B14 */

```

```

items_in_bag(Bag_number,Size,Number) :-
    initiated_bag(Bag_number,Item_list),
    count_items(Item_list,Size,Number).

```

```

count_items([],Size,0).
count_items([Head|Tail],Size,Number) :-
    size(Head,Size),
    count_items(Tail,Size,N),
    Number is N + 1.

```

```

count_items([Head|Tail],Size,Number) :-
    count_items(Tail,Size,Number).

```

```

out_item_in_bag(Item,Bag_number) :-
    retract(initiated_bag(Bag_number,Item_list)),
    append(Item_list,[Item],New_item_list),
    assertz(initiated_bag(Bag_number,New_item_list)),
    retract(unbagged(Item)),
    printstring(" < "),
    print(Item),
    printstring(" > is placed in bag number < "),
    print(Bag_number),
    printstring(" >"),
    nl.

```

```

initiated_bag(1,[]).
number_of_bags(1).

```

```

start_fresh_bag :-
    retract(number_of_bags(N)),
    M is N + 1,
    assertz(number_of_bags(M)),
    assertz(initiated_bag(M,[])).

```

```

empty_bag_or_bag_with_medium_items(Bag_number) :-
    initiated_bag(Bag_number,[]).

```

```

empty_bag_or_bag_with_medium_items(Bag_number) :-
    initiated_bag(Bag_number,Itemlist),
    count_items(Itemlist,medium,Number),
    Number > 0.

```

```
contains(Bag_number,Container) :-
    initiated_bag(Bag_number,Item_list),
    container(Item,Container),
    member(Item,Item_list).

put_in_insulated_freezer_bag(Item) :-
    assertz(insulated(Item)),
    printstring(" < "),
    print(Item),
    printstring(" > is put in an insulated freezer bag"),rl.

not_full_bag(Bag_number) :- /* the check is not implemented,
                             /* every bag is consider to be not_full
    initiated_bag(Bag_number,Item_list).
```

yes

| ?-

[ Prolog execution halted ]

;

script done on Sat Jan 5 18:55:59 1985